凝聚名家技术典范·分享成功IT之路



DB2高级管理、系统设计与诊断案例(第3版)



牛新庄　著

清华大学出版社

DB2 高级管理、系统设计与诊断案例

(第 3 版)

牛新庄　著

清华大学出版社

北　　京

内 容 简 介

数据库内核是数据库系统稳定运行的心脏，DB2 数据库内部结构庞大而复杂。本书从 DB2 进程和内部线程结构入手，介绍代理程序工作机制、内存体系结构、存储体系结构等。在此基础上详解了 DB2 数据库的高级功能，包括数据分区、高级压缩功能、安全特性等，并系统地介绍了 OLTP 和 OLAP 系统的设计方案和管理技术、高可用和容灾方案以及集群技术，其中包含了 HADR、DPF 和 pureScale 技术，以及同城双活 GDPC(地理上分离的 pureScale 集群)技术。还介绍了 DB2 各种监控和诊断方法，通过精选的诊断案例使读者在学习知识的同时积累了实践经验。在新的一版中，所有的内容、示例都基于 DB2 V10.5 版本进行了修订。

本书适合具有一定 DB2 基础知识和经验的数据库工程师，非常适合希望能了解 DB2 的内部结构、提高各种故障的诊断和调优的能力、想成为资深 DB2 数据库工程师的读者。

图书在版编目(CIP)数据

DB2 高级管理、系统设计与诊断案例 / 牛新庄 著. —3 版. —北京：清华大学出版社，2017
ISBN 978-7-302-48117-1

Ⅰ. ①D… Ⅱ. ①牛… Ⅲ. ①关系数据库系统—研究 Ⅳ. ①TP311.138

中国版本图书馆 CIP 数据核字(2017)第 208033 号

责任编辑： 王　军　李维杰
装帧设计： 牛艳敏
责任校对： 成凤进
责任印制：

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　**邮　　编：** 100084
社 总 机： 010-62770175　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185mm×230mm　**印　张：** 29.25　**字　数：** 602 千字
版　　次： 2009 年 5 月第 1 版　2017 年 9 月第 3 版　**印　次：** 2017 年 9 月第 1 次印刷
印　　数： 1~3000
定　　价： 98.00 元

产品编号：

序

关系型数据库已经走过整整半个世纪曲折而辉煌的历程，回顾 50 年的关系数据库发展史，我们心潮涌动，激情难抑。关系数据库始于 1970 年 IBM 公司研究员 E.F.Codd 博士，即“关系数据库之父”，发表的业界第一篇关于关系数据库理论的论文“A Relational Model of Data for Large Shared Data Banks”。从此关系型数据库如雨后春笋般四处萌发，各大厂商争先恐后，加入到关系型数据库的研发大潮中，而后又如大浪淘沙般去璞存真，时至今日，留下的是真正适用于客户、适应于潮流的关系型数据库产品。

IBM 公司作为关系数据库的推广先锋，为业界提供了一批优秀的数据库技术领域先驱科学家，他们所研发出的 DB2 数据库，经过近 50 年的发展，已经广泛应用于金融、电信、制造等多个行业，对日常的工作和生活带来了深远的影响。

在中国，DB2 数据库的兴起大概在 15 年前，当时国内鲜有应用，牛新庄先生作为国内首批接触研究 DB2 数据库的工程师，为业界提供了大量技术服务和专业培训，为 DB2 数据库的推广应用做出了积极杰出的贡献。同时，也为 IBM 公司反馈了很多很好的 DB2 研发建议，为关系型数据技术的长远发展贡献智慧。在数据库方向的精深造诣和丰富实践，就浓缩在他的 DB2 数据库著作中，为广大 IT 同仁授业解惑。

本套图书可以说是伴随着 DB2 数据库的成长，从第一版主讲 DB2 V8，到第二版的 DB2 V9，再到第三版的 DB2 V10。基本上每出一个版本我都会仔细品味，每个版本作者都很用心，都会删减对当前不适用的章节，加入很多新的功能和他近期实际经历的案例。经典性、权威性、实用性是本套书籍锁定的主要目标。

第一，本套书籍涵盖了 DB2 的几乎所有功能，是业界最大规模的系统梳理与总结。从理论知识到最佳实践，无所不包，无所不有，同时还总结了几十个最佳案例。

第二，本套书籍理论讲解深入浅出，案例多样详实，无论是对零基础还是拥有多年 DBA 工作经验的人都非常适用。

第三，精品的价值在于传世久远，经典的意义在于常读常新。我认为，只有被广泛阅读，受到大家喜爱、接受的作品，才具有经典的资质与意义。希望作者继续努力，将本套书籍打磨成关系数据库的经典图书。

其中第一本书《循序渐进 DB2　DBA 系统管理、运维与应用案例(第 3 版)》，是 DB2 学习的入门书籍。该书包含了从入门到中级阶段的知识和技能的介绍，全面展示了 DB2 的主要功能和日常的工作技巧，尤其是实例和案例部分，这部分内容从实际出发为读者列举了常见的案例场景和处理办法，非常实用。在新的一版中，所有的内容、示例都基于 DB2 V10.5 版本进行了修订，并介绍了 DB2 V11.1 中的新功能、新特性。

第二本书《DB2 高级管理、系统设计与诊断案例(第 3 版)》是 DB2 学习的高级进阶，该书从 DB2 体系结构入手，介绍了 DB2 各个内部组件的层次与功能、内存内部结构、存储内部结构、锁和并发原理等。在理解 DB2 内部基本原理的基础上进一步介绍了 DB2 的高级功能，包括分区功能、高级压缩功能等。此外，系统介绍了 OLTP 和 OLAP 系统的设计方法和管理技术、高可用和容灾方案，以及集群技术，其中涉及 HADR、DPF 和 pureScale 技术，以及地理上分离的 pureScale 集群(GDPC)技术。该书还介绍了 DB2 各种监控和诊断方法，通过精选的诊断案例使读者在学习知识的同时积累实践经验。

第三本书《DB2 数据库性能调整和优化(第 3 版)》专门介绍 DB2 性能调整和优化，从 DB2 数据库性能有关的基础知识和原理入手，从数据库所处的运行环境(OS、存储等)开始介绍，并对 DB2 的进程和内存进行深入讲解。在全面了解性能相关知识后，开始逐步展开，从设计到监控，从配置参数到调优工具，从锁和并发到优化器统计信息，最后列出了几个完整的性能调优案例以增加技术理解。

祝愿每一位读者能有所得、有所悟，成长为新一代的数据技术专家，也祝愿牛新庄先生在数据技术领域这条康庄大道上走得更宽更远。

IBM 前大中华区总经理
IBM 大中华区高级顾问　　王天義

前　　言

自 1999 年左右我开始从事数据库有关的技术工作到现在已近 20 年时间，此期间信息科技飞速发展，从无纸化办公和数据大集中到移动互联和大数据、人工智能、云计算等信息技术改变了生活，并颠覆了传统商业模式。信息科技的发展离不开数据处理技术的进步，在这一轮信息化浪潮中，数据处理技术也发生了翻天覆地的变化，对企业经营发展和对外服务的意义越来越重要。一方面，传统企业级数据库的能力，在原有的道路和方向上不断地持续提升演进，以满足企业市场不断迸发的各类需求。另一方面，互联网场景孕育的各种新兴的数据处理技术亦不断涌现，例如 NewSQL、NoSQL、Hadoop 等大数据处理技术，这些技术成为传统数据库产品的必然补充，同时也对传统数据库产品产生了一定的冲击。但是以我长期从事企业数据处理相关工作的经验看，在企业级市场尤其是金融企业市场里面，传统数据库产品的能力依然是解决企业主要业务需求的不二选择。因此，传统数据库技术的研究和应用仍然是信息科技工作的重点。

近年来传统数据库产品在不断改进升级，以支持更快的处理能力和更高的可用性，满足不同场景下的用户需求。DB2 作为一款主流数据库产品，在这些方面也都进步明显，例如 Purescale 集群技术、跨数据中心的 GDPC 技术、列存储的 BLU 技术等创新功能就表现不俗，满足了特定业务场景需求，给企业带来了很大的价值提升。特别是 GDPC 技术，帮助企业搭建关键业务系统同城对等全双活生产架构，为最终用户提供高等级容灾的连续服务，对企业对外服务的提升意义非凡，也使数据库从业者们领略了 DB2 产品创新的精华。

基于 DB2 产品的演进以及近些年的思考和实践，我重新梳理了之前编写的第 2 版的 3 本 DB2 系列技术图书，对其进行了大篇幅的修改和重写，力图对近些年实践的精华和 DB2

产品的新趋势进行总结。在此奉献给各位数据库从业的同仁，在技术的路上共勉。

由于本人水平有限，时间有限，书中不免有这样或者那样的错误，希望广大读者朋友不吝赐教指正！

最后，感谢我的家人和同事在本书重写过程中的帮助，谢谢你们！

牛新庄

目　录

第 1 章 DB2 体系结构

无论是系统软件还是应用软件，都离不开进程和内存这两种体系结构，作为数据库管理软件的 DB2 也不例外。这两种结构关系到整个软件的运行基础，作为 DB2 DBA 有必要详细深入了解 DB2 的进程和内存体系结构，这对 DB2 的运维和管理起着至关重要的作用。

在应用系统的开发设计前期，如果数据库的物理设计和逻辑设计合理，整个应用系统的基础架构将非常坚实，国内很多应用系统往往是前期设计不合理才导致后期花费很多时间和精力来进行调整的。所以，良好的数据库物理设计和逻辑设计是实现系统高效运行的第一步。在本章，我们将讲解 DB2 的进程体系结构、内存体系结构和存储体系结构，以及如何针对自己的业务特点来选择最合理的数据库物理设计和逻辑设计。

1.1 DB2 进程体系结构

理解 DB2 的进程体系结构有助于监控数据库的内部活动以及调优，从而获得更好的性能。下面我们就针对 DB2 的进程和内存体系结构逐一深入介绍。

DB2 进程体系结构涉及如下内容：

- DB2 进程技术模型
- 代理进程通信
- 实用程序相关进程

1.1.1 DB2 进程技术模型

DB2 进程技术模型方面的知识可以帮助您理解数据库管理器与其相关联的组件的交互方式，并且可以帮助您在发生问题时进行故障诊断。所有 DB2 数据库服务器使用的进程技术模型都旨在简化数据库服务器与客户机之间的通信，此外还确保数据库应用程序独立于数据库控制块和关键数据库文件之类的资源。DB2 数据库服务器必须执行各种不同的任务，例如处理数据库应用程序请求或确保将日志记录写入磁盘。通常，每项任务都由独立的引擎可分派单元(EDU)执行。

采用多线程体系结构对于 DB2 数据库服务器而言有很多优点。由于同一进程内的所有线程可以共享一些操作系统资源，因此新线程需要的内存和操作系统资源要比进程少。此外，在多数平台上，线程的上下文切换时间要比进程短，这有助于提高性能。在所有平台上使用线程模型使得 DB2 数据库服务器更易于配置，因为这样更容易根据需要分配更多 EDU，并且可以动态分配必须由多个 EDU 共享的内存。

对于正在访问的每个数据库，将启动不同的 EDU 以处理各种数据库任务，例如预取、通信和日志记录。数据库代理程序是一类特殊的 EDU，创建它们是为了处理应用程序对数据库的请求。

通常，可以依靠 DB2 数据库服务器来管理 EDU 集合。但是，也可以通过一些 DB2 工具来管理 EDU。例如，可以使用 db2pd -edus 命令来列示所有活动的 EDU 线程。

每个客户机应用程序连接都有一个对数据库执行操作的协调代理程序。协调代理程序代表应用程序工作，并根据需要使用专用内存、进程间通信(IPC)或远程通信协议与其他代理程序进行通信。

DB2 体系结构提供了防火墙，以使应用程序与 DB2 数据库服务器在不同的地址空间中运行。防火墙将数据库和数据库管理器与应用程序、存储过程和用户定义函数(UDF)隔开。防火墙有助于维护数据库中数据的完整性，这是因为防火墙能阻止应用程序编程错误覆盖内部缓冲区或数据库管理器文件。防火墙还提高了可靠性，原因是应用程序错误不会导致数据库管理器崩溃。

图 1-1 是 DB2 进程体系结构的技术模型图。

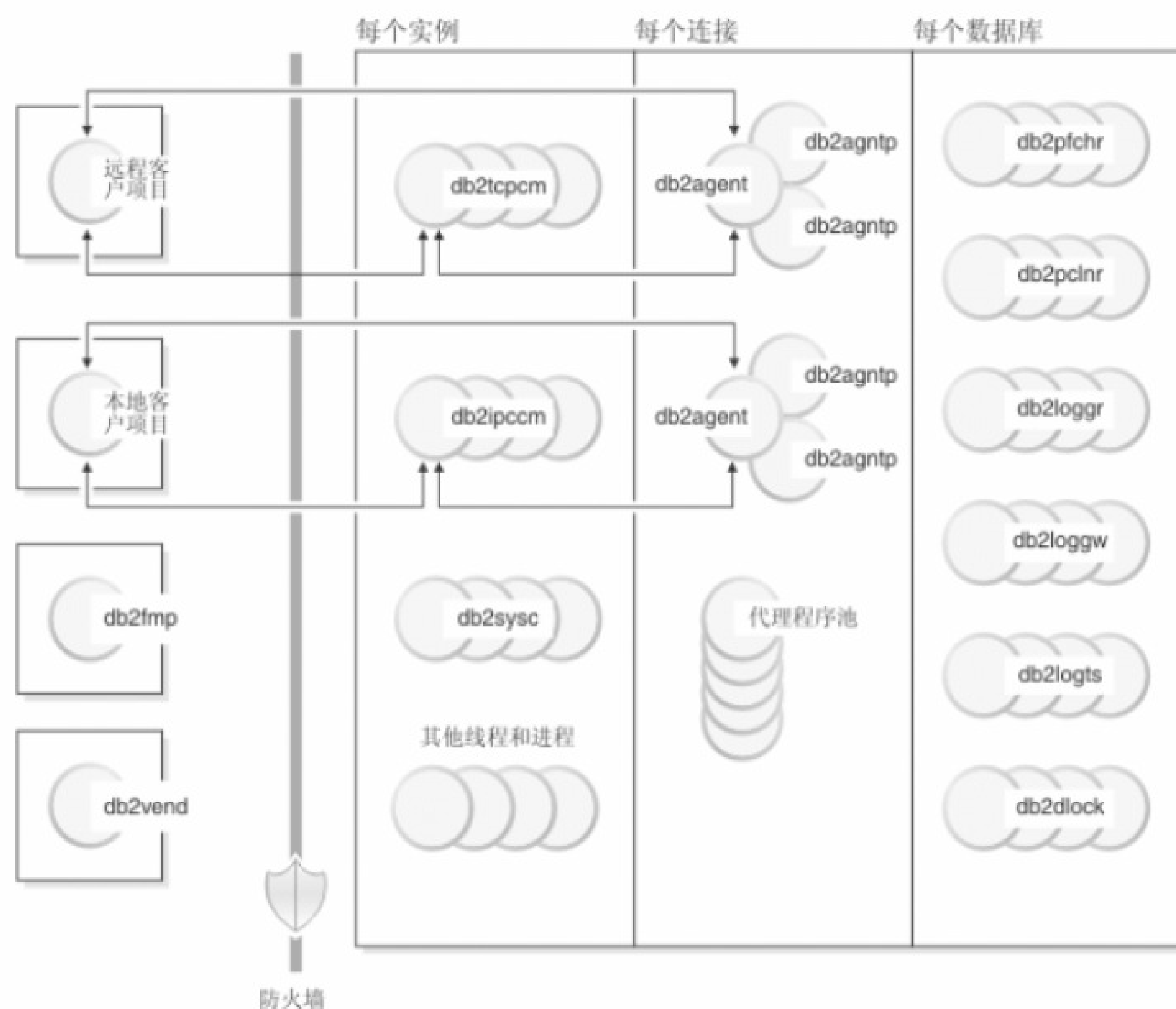


图 1-1　DB2 进程技术模型图

下面我们详细介绍 DB2 进程技术模型中的相关进程。

1.1.2　与操作系统相关的进程

我们都知道 DB2 是安装在操作系统上的，那么在我们使用 DB2 数据库时，会产生一些和操作系统相关的进程，通常这些进程的拥有者是操作系统的 root 用户。下面是在 AIX 上使用“ps –ef |grep db2”命令格式化之后的输出结果，如下所示：

```
$ ps -ef |grep db2
     UID      PID      PPID          CMD
    root  7012568         1      /opt/IBM/db2/v9.7.5/bin/db2fmcd
db2inst1 12058724 15400994   db2fmp (C) 0
db2inst1 13500634 15400994   db2acd 0
db2inst1 13697222 14680224   db2vend (PD Vendor Process - 258)
    root 13959272 14680224   db2ckpwd 0
db2inst1 14024834 15400994   db2fmp (idle) 0
    root 14352626 14680224   db2ckpwd 0
```

```
db2inst1 14680224 15400994   db2sysc 0
    root 14942460 14680224    db2ckpwd 0
    root 15400994        1     db2wdog 0
```

从命令的输出中我们可以看到："db2wdog"、"db2ckpwd" 和 "db2fmcd" 进程的拥有者都是操作系统的 root 用户。下面我们详细讲解这些和操作系统相关的进程。

1. db2wdog 进程

我们都知道在 UNIX/Linux 上，init 进程是所有进程的父进程；同样，在 DB2 进程中，"db2wdog" 进程是所有其他 DB2 进程的父进程。这个进程是由操作系统的 init 进程派生的。从上面的命令输出中我们可以看到："db2wdog" 的 "ppid" (parent pid)是操作系统的 init 进程。"db2wdog" 进程是在 UNIX 和 Linux 操作系统上处理异常终止的看守程序。

"db2wdog" 是 "db2 watch dog" 的缩写，是看门狗的意思，只要有 DB2 进程接收到 CTRL-C 或其他异常信号，该进程就会向看守程序发送信号，而看守程序会将信号传播给实例中的其他所有进程。如果该进程出现异常，那么整个 DB2 数据库将无法正常工作。

2. db2ckpwd 进程

DB2 使用的用户只能是操作系统用户，DB2 使用的安全机制则依赖操作系统或第三方安全插件来实现。那么如果有应用程序连接数据库，如何验证用户名和密码的合法性呢？"db2ckpwd" 进程用于检查 DB2 服务器上的用户标识和密码。由于 DB2 依赖于操作系统级别的认证，因此当某个用户或应用程序连接到服务器上的数据库时，使用该进程验证用户标识和密码。当将 AUTHENTICATION 设置为 SERVER 时，或者当连接是从非安全的操作系统建立时，就会进行认证。

图 1-2 显示了利用 "db2ckpwd" 进程验证用户和密码的过程。

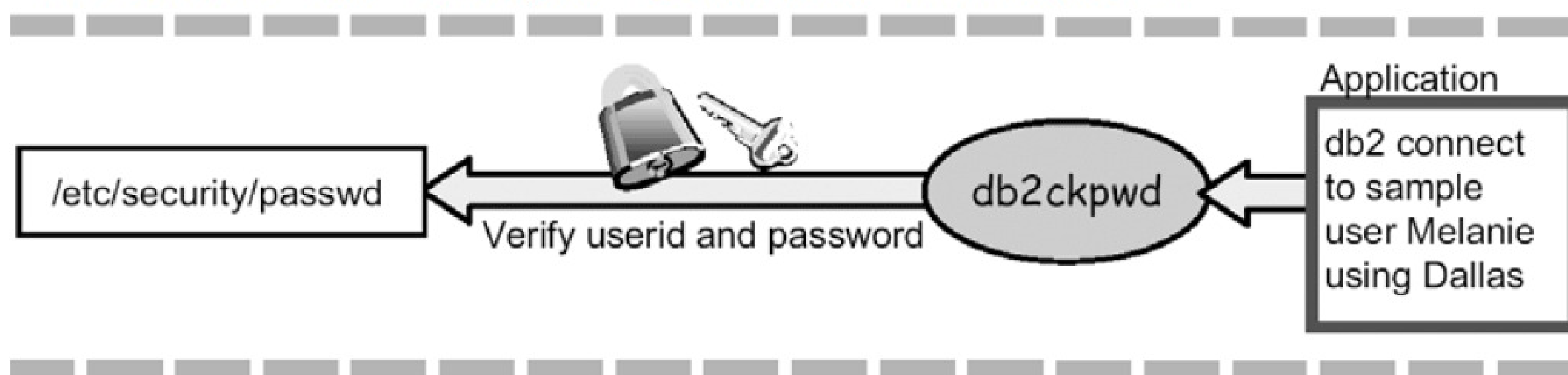


图 1-2　db2ckpwd 进程的工作机制

在图 1-2 中，当用户发出 "db2 connect to sample user Melanie using Dallas" 命令后，

“db2ckpwd”进程把用户名“Melance”和密码“Dallas”与“/etc/security/passwd”文件中相应的内容进行比较，以验证用户和密码的合法性。

3. db2fmcd 进程

该进程是 DB2 的故障监视器进程，当实例宕机后，它会自动重启实例。

1.1.3　与实例相关的进程和线程

当我们执行 db2start 时，会在操作系统上产生进程 db2sysc；每个实例都有自己独立的进程，这些进程的名字是一样的，但是进程的拥有者属于不同的实例，同时还会派生出和实例相关的一些线程，可以通过 db2pd -edus 命令来查看，输出如下：

```
$db2pd -edus
Database Partition 0 -- Active -- Up 0 days 00:00:06 -- Date 2013-01-29-15.11.07.762341
List of all EDUs for database partition 0
db2sysc PID: 55115990
db2wdog PID: 41353298
db2acd  PID: 43122742

EDU ID    TID          Kernel TID     EDU Name                        USR (s)      SYS (s)
===========================================================================================
3856      3856         41222273       db2spmlw 0                      0.000066     0.000253
3599      3599         27066371       db2spmrsy 0                     0.002758     0.000438
3342      3342         65732611       db2resync 0                     0.000216     0.000135
3085      3085         44892197       db2tcpcm 0                      0.000058     0.000013
2828      2828         62652665       db2tcpcm 0                      0.000072     0.000013
2571      2571         21758109       db2tcpcm 0                      0.000056     0.000012
2314      2314         59310105       db2tcpcm 0                      0.000048     0.000012
2057      2057         64356517       db2tcpcm 0                      0.000051     0.000013
1800      1800         46399679       db2tcpcm 0                      0.000074     0.000012
1543      1543         71106699       db2tcpcm 0                      0.000060     0.000012
1286      1286         81920135       db2tcpcm 0                      0.000079     0.000018
1029      1029         68288767       db2ipccm 0                      0.000071     0.000058
772       772          70254661       db2licc 0                       0.000100     0.000194
515       515          54001737       db2thcln 0                      0.000073     0.000020
2         2            78250139       db2alarm 0                      0.000114     0.000049
258       258          42795019       db2sysc 0                       0.016676     0.008767
```

这些只是和实例相关的部分线程，和实例相关的更多线程的列表如下：

- db2acd，用于主管运行状况监视器、自动维护实用程序和管理任务调度程序的自主计算守护程序。此进程以前称为 db2hmon。
- db2aiothr，用于管理数据库分区的异步 I/O 请求(仅适用于 UNIX)。
- db2alarm，用于在 EDU 请求的计时器到期时通知 EDU(仅适用于 UNIX)。
- db2cart，用于在 userexit 数据库配置参数处于启用状态时归档日志文件。
- db2disp，客户机连接集中器分派器。
- db2fcms，快速通信管理器发送方守护程序。
- db2fcmr，快速通信管理器接收方守护程序。
- db2fmd，故障监视器守护程序。
- db2fmtlg，用于在 logretain 数据库配置参数处于启用状态并且 userexit 数据库配置参数处于禁用状态时格式化日志文件。

- db2licc，管理已安装的 DB2 许可证。
- db2panic，紧急代理程序，用于在特定数据库分区达到代理程序的限制时处理紧急请求(仅用于分区数据库环境)。
- db2pdbc，并行系统控制器，用来处理来自远程数据库分区的并行请求(仅用于分区数据库环境)。
- db2resync，扫描全局再同步列表的再同步代理进程。
- db2sysc，主系统控制器 EDU，用于处理关键的 DB2 服务器事件。
- db2thcln，在 EDU 终止时重新启动资源(仅适用于 UNIX)。
- db2wdog，在 UNIX 和 Linux 操作系统上处理异常终止的看守程序。

1.1.4 与数据库相关的进程和线程

当我们第一次连接数据库或执行“activate db”命令时，会启动数据库相关的线程，如 db2pfchr、db2pclnr、db2loggr、db2loggw 等，可以通过 db2pd -edus 命令来查看，输出如下：

```
$db2pd -edus
Database Partition 0 -- Active -- Up 0 days 00:03:25 -- Date 2013-01-29-15.14.26.558482
List of all EDUs for database partition 0
db2sysc PID: 55115990
db2wdog PID: 41353298
db2acd  PID: 43122742

EDU ID    TID        Kernel TID    EDU Name                        USR (s)     SYS (s)
=====================================================================================
11824     11824      72548545      db2evmti (DEAD) 0               0.001601    0.001830
11567     11567      43122799      db2evmgi (DB2DETAILDEADLOCK) 0  0.000709    0.000698
9768      9768       65863819      db2fw1 (SAMPLE) 0               0.000644    0.000543
9511      9511       51380301      db2fw0 (SAMPLE) 0               0.000715    0.000549
9254      9254       42270743      db2lused (SAMPLE) 0             0.000911    0.000630
8997      8997       76546293      db2wlmd (SAMPLE) 0              0.000581    0.000491
8740      8740       84738073      db2taskd (SAMPLE) 0             0.000975    0.001161
8483      8483       85000441      db2stmm (SAMPLE) 0              0.001541    0.001564
7969      7969       73400393      db2pfchr (SAMPLE) 0             0.000103    0.000010
7712      7712       65798151      db2pfchr (SAMPLE) 0             0.000097    0.000016
6170      6170       46727179      db2pclnr (SAMPLE) 0             0.000082    0.000015
5913      5913       31391857      db2pclnr (SAMPLE) 0             0.000094    0.000019
5656      5656       70909971      db2dlock (SAMPLE) 0             0.000101    0.000026
5399      5399       87032027      db2lfr (SAMPLE) 0               0.000044    0.000009
5142      5142       79036477      db2loggw (SAMPLE) 0             0.000904    0.001059
4885      4885       89194699      db2loggr (SAMPLE) 0             0.001003    0.002718
4628      4628       39780361      db2logmgr (SAMPLE) 0            0.000108    0.000040
4371      4371       36962555      db2logts (SAMPLE) 0             0.000118    0.000022
4114      4114       69533791      db2agent (SAMPLE) 0             0.028280    0.024763
3856      3856       41222273      db2spmlw 0                      0.000066    0.000253
3599      3599       27066371      db2spmrsy 0                     0.002764    0.000449
3342      3342       65732611      db2resync 0                     0.000219    0.000146
3085      3085       44892197      db2tcpcm 0                      0.000058    0.000013
2828      2828       62652665      db2tcpcm 0                      0.000072    0.000013
2571      2571       21758109      db2tcpcm 0                      0.000056    0.000012
2314      2314       59310105      db2tcpcm 0                      0.000048    0.000012
2057      2057       64356517      db2tcpcm 0                      0.000051    0.000013
1800      1800       46399679      db2tcpcm 0                      0.000074    0.000012
1543      1543       71106699      db2tcpcm 0                      0.000060    0.000012
1286      1286       81920135      db2tcpcm 0                      0.000079    0.000018
1029      1029       68288767      db2ipccm 0                      0.000194    0.000149
772       772        70254661      db2licc 0                       0.000100    0.000194
515       515        54001737      db2thcln 0                      0.000091    0.000025
2         2          78250139      db2alarm 0                      0.001155    0.000948
258       258        42795019      db2sysc 0                       0.034475    0.032803
```

以下列表包括每个数据库都会使用的一些重要线程：

- db2dlock，用于死锁检测。在分区数据库环境中，使用另一个线程(db2glock)来协调 db2dlock EDU 从每个分区中收集的信息；db2glock 仅对目录分区运行。
- db2fw，事件监视器快速写程序；用于对表、文件或管道进行事件监视器数据的大量、并行写入。
- db2hadrp，高可用性灾难恢复(HADR)主服务器线程。
- db2hadrs，HADR 备用服务器线程。
- db2lfr，用于处理各个日志文件的日志文件阅读器。
- db2loggr，用于处理日志文件以处理事务处理和恢复。
- db2loggw，用于将日志记录写入日志文件。
- db2logmgr，用于日志管理器，管理可恢复数据库的日志文件。
- db2logts，用于跟踪哪些表空间在哪些日志文件中有日志记录。此信息记录在数据库目录的 DB2TSCHG.HIS 文件中。
- db2lused，用于更新对象用途。
- db2pfchr，用于缓冲池预取程序。
- db2pclnr，用于缓冲池页清除程序。
- db2redom，用于重做主进程。在恢复期间，处理重做日志记录并将日志记录指定给重做工作程序来进行处理。
- db2redow，用于重做工作程序。在恢复期间，按照重做主进程的请求来处理重做日志记录。
- db2shred，用于处理日志页中的各个日志记录。
- db2stmm，用于自调整内存管理功能。
- db2taskd，用于分发后台数据库任务，这些任务由名为 db2taskp 的线程执行。
- db2wlmd，用于自动收集工作负载管理统计信息。

事件监视器线程的标识方式如下：

db2evm*%1%2*(*%3*)

其中，*%1* 可以是：

- g——全局文件事件监视器
- gp——全局管道事件监视器
- l——本地文件事件监视器
- lp——本地管道事件监视器
- t——表事件监视器

%2 可以是：

- i——协调程序

- p——非协调程序

而 %3 是事件监视器名称。

备份和复原线程的标识方式如下：

db2bm.%1.%2(备份和复原缓冲区操纵程序)和 db2med.%1.%2(备份和复原介质控制器)，其中：%1 是用于控制备份或复原会话的代理程序的 EDU 标识，%2 是用于区分属于特定备份或复原会话的线程(可能有许多个)的顺序值。例如，db2bm.13579.2 标识具有 EDU 标识为 13579 的 db2agent 线程控制的第二个 db2bm 线程。

在数据库中，有如下 3 个线程与磁盘 I/O 活动密切相关，它们对数据库的性能有着重要影响：

- **预取程序**(db2pfchr)在应用程序需要数据之前从磁盘检索数据，并将数据移到缓冲池中。例如，如果没有数据预取程序，那么需要扫描大量数据的应用程序必须等待数据从磁盘移到缓冲池中。应用程序的代理程序将异步预读取请求发送至公共预取队列。当预取程序可用时，它们使用大块或随机读取输入操作来将请求的页从磁盘读入缓冲池，从而实现那些请求。如果具有多个磁盘来存储数据库数据，那么可以将数据条带分割到多个磁盘上。条带分割数据让预取程序同时使用多个磁盘来检索数据。
- **页清除程序**(db2pclnr)将数据从缓冲池移动到磁盘中。页清除程序是独立于应用程序代理程序的后台 EDU。它们将查找已经修改的脏页，并将已更改的那些脏页写入磁盘。页清除程序确保缓冲池中有空闲空间供预取程序正在检索的页使用。
- **日志写入程序**(db2loggw)将日志缓冲区的日志记录写入日志文件。

这 3 个进程的工作原理如图 1-3 所示。

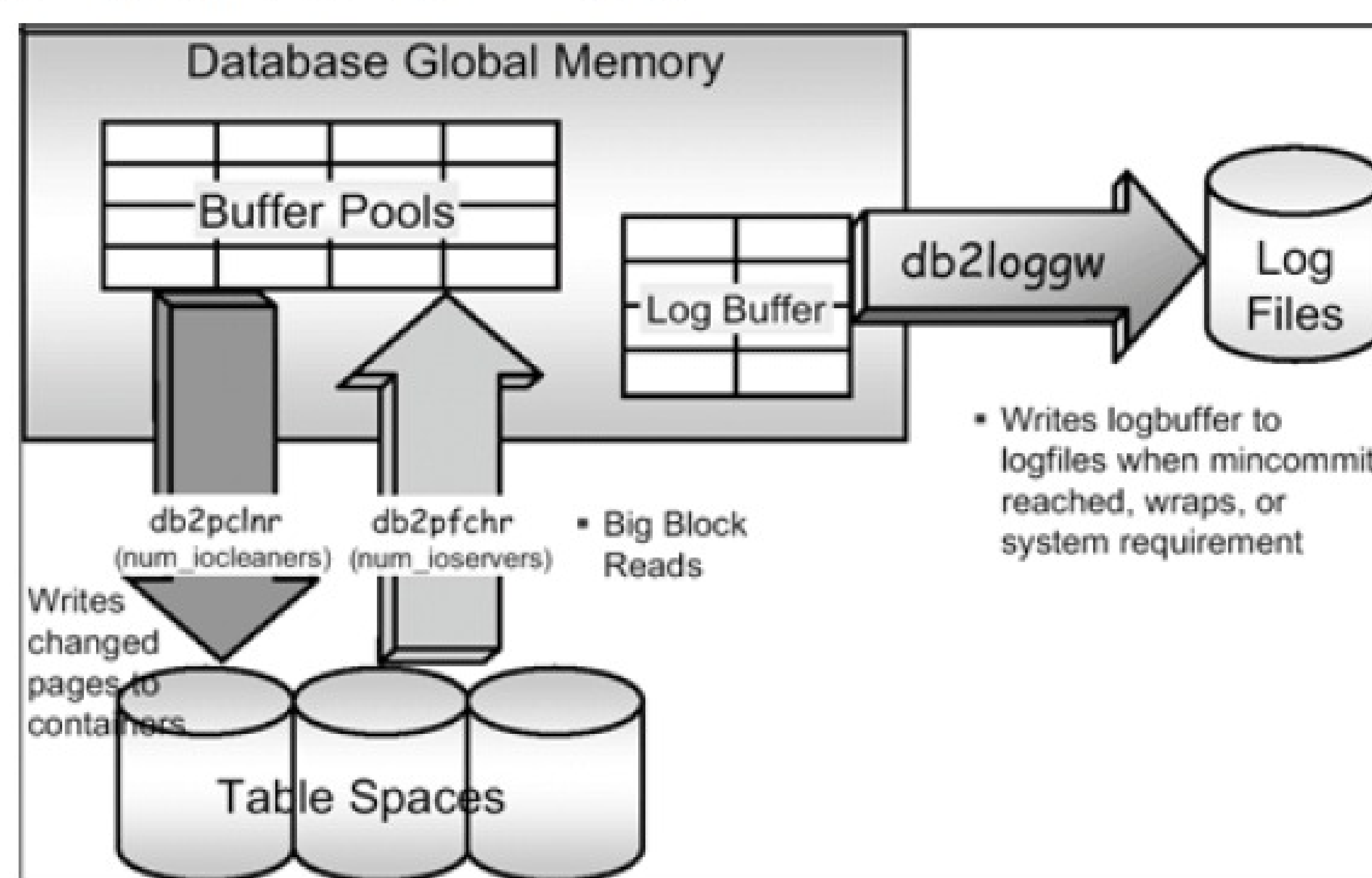


图 1-3　DB2 中与 I/O 相关的进程

如果没有独立的预取程序和页清除程序 EDU，那么应用程序的代理程序必须自己执行缓冲池与磁盘存储器之间的所有数据读取和写入操作，这会严重影响数据库的性能。

在数据库中我们可以通过配置参数 num_iocleaners 和 num_ioservers 来限制线程 db2pclnr 和 db2pfchr 的个数，虽然推荐的是把这两个参数的值设成 automatic，但是在实际的运维中，automatic 并不能很好地发挥作用，建议从 30 起进行设置。

1.1.5　与应用程序相关的进程

当我们启动了实例和数据库后，应用程序就可以连接数据库进行相关操作了。每个应用程序在连接数据库时都会产生 db2agent 的 EDU，在 DB2 进程技术模型中，和应用相关的进程如图 1-4 所示。

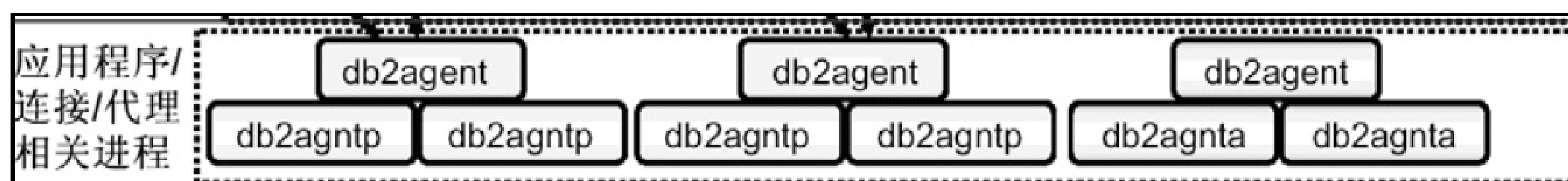


图 1-4　与应用程序相关的进程模型图

当应用程序连接数据库时，DB2 会对来自客户机应用程序的所有连接请求(不管是本地的还是远程的)分配相应的协调代理程序(“db2agent”)。协调代理程序将代表应用程序执行所有的数据库请求。

在启用了数据库分区功能部件(DPF)或 INTRA_PARALLEL 并行性的环境中，协调代理程序会将数据库请求分发给子代理程序(名称分别为“db2agntp”和“db2agnts”)。这些代理程序为应用程序执行请求。一旦创建协调代理程序，就会代表应用程序处理所有数据库请求，方法是协调对数据库执行请求的子代理程序(“db2agntp”)。与应用程序关联但当前处于空闲状态的子代理程序用名称“db2agnta”标识。

协调代理程序可能：

- 连接至具有别名的数据库。例如，“db2agent(SAMPLE)”会连接至数据库别名“SAMPLE”。
- 连接至实例。例如，“db2agent(DB2INST1)”会连接至实例“DB2INST1”。

DB2 进程技术模型将实例化其他类型的代理程序以执行特定操作，如独立的协调代理程序或子协调代理程序。例如，独立的协调代理程序“db2agnti”用于运行事件监视器，而子协调代理程序“db2agnsc”用于在突然关闭后以并行方式执行数据库重新启动。

空闲代理程序位于代理程序池中。这些代理程序对于来自代表客户机程序运行的协调代理程序或来自代表现有协调代理程序运行的子代理程序的请求是可用的。在涉及大量应用程序工作负载的配置中具有大小合适的空闲代理程序池有助于提高性能，这是因为可以根据需要立即使用空闲代理程序，而不必为每个应用程序连接分配全新的代理程序，这涉及创建线程以及分配并初始化内存和其他资源。从 V9.5 开始，DB2 还可以在需要时动态管理空闲代理程序池的大小。

DB2 中和代理程序相关的进程列表如表 1-1 所示。

表 1-1 DB2 中常见的和代理程序相关的进程及描述

进 程 名	描 述
db2agent	DB2 协调程序代理程序，代表应用程序执行所有数据库请求。除非启用连接集中器，否则每个已连接的应用程序都将有一个 db2agent 进程 如果启用了分区内并行性，那么 db2agent 进程将调用 DB2 子代理程序来执行工作，并且它们会将结果集返回给协调程序代理程序，然后再返回给应用程序 在分区数据库中，协调程序代理程序将位于应用程序连接到的分区上
db2agentg	DRDA 应用程序请求器(Application Requester)的网关代理程序
db2agnsc	并行恢复代理程序。在前滚和重新启动恢复的过程中使用该代理程序来并行地执行日志中的操作。与串行恢复相比，这可以缩短恢复时间 注意：该进程支持已记录事务中的并行性以及并行事务之间的并行性
db2agnta	空闲的子代理程序，过去协调代理曾使用过，并且现在仍然与协调代理进程关联 当 INTRA_PARALLEL dbm cfg 参数设置成 YES 时会出现该进程
db2agntp	这是子代理程序，代表与之相关的协调代理执行当前工作。这些进程提供了分区内并行性，也就是在数据库实例/分区中并行地执行查询的能力。当 INTRA_PARALLEL dbm cfg 参数设置成 YES 时会出现该进程

1.1.6 监控 EDU 运行的 SQL 语句

执行“db2pd -edu”，在下面的输出中找出 USR 和 SYS 列中最大，也就是最占用 CPU 的 EDU ID，输出结果如下：

```
$ db2pd -edu
Database Partition 0 -- Active -- Up 1 days 01:05:54
List of all EDUs for database partition 0
db2sysc PID: 1921136
db2wdog PID: 2109662
db2acd  PID: 1237176
EDU ID    TID      Kernel TID    EDU Name               USR          SYS
==========================================================================
3560      3560     2216003       db2agent (SAMPLE)      24.706935    1.071737
1543      1543     5628153       db2resync 0            0.002641     0.004271
1286      1286     1851457       db2ipccm 0             0.082388     0.044037
1029      1029     2023571       db2licc 0              0.000211     0.001055
772       772      4390991       db2thcln 0             0.000244     0.000105
515       515      2621455       db2aiothr 0            2.740874     6.287562
2         2        1273899       db2alarm 0             0.274076     0.408226
```

```
    258        258      2658531        db2sysc 0              2.085981      1.379128
```

根据这个 EDU ID 执行“db2pd –db sample –application”，找出和这个 EUD ID 相关联的应用程序句柄(APPHANDL)，输出结果如下：

```
    $ db2pd -db sample -applications
    Database Partition 0 -- Database SAMPLE -- Active -- Up 0 days 00:49:26
    Applications:
    Address    AppHandl [nod-index] NumAgents  CoorEDUID  Status  C-AnchID
C-StmtUID  L-AnchID L-StmtUID  Appid  WorkloadID  WorkloadOccID
    0x7AF02EC0 3361     [000-03361] 1         1372      ConnectCompleted
0       0         0       0         *LOCAL.DB2 01.09010409
    1922                                   0         0
    0x7AF00060 3360     [000-03360] 1         5920      ConnectCompleted
0       0         0       0         *LOCAL.DB2.09010409192
    1                                     0         0
    0x7AEFA8B0 3359     [000-03359] 1         2548      ConnectCompleted
0       0         0       0         *LOCAL.DB2.09010409192
    0                                     0         0
    0x7AEF7A50 3358     [000-03358] 1         3808      ConnectCompleted
0       0         0       0         *LOCAL.DB2.09010409191
    9                                     0         0
    0x7AEF4BF0 3357     [000-03357] 1         5144      ConnectCompleted
0       0         0       0         *LOCAL.DB2.09010409191
    8                                     0         0
    0x7AEDAEE0 3356     [000-03356] 1         3560      UOW-Waiting
0       0         190      1         *LOCAL.DB2_01.09010409
    1915                                   1         1
    ......................................................略.....................................
```

根据这个应用程序句柄，然后执行“db2 get snapshot for application agentid <applhandle>”即可定位最占用 CPU 资源的 SQL 语句。输出结果如下所示：

```
    $ db2 get snapshot for application agentid 3356
    ......................................略.............................................
    Buffer pool data logical reads             = 0
    Buffer pool data physical reads            = 0
    Buffer pool temporary data logical reads    = 0
    Buffer pool temporary data physical reads  = 0
    Buffer pool index logical reads            = 0
    Buffer pool index physical reads           = 0
    Buffer pool temporary index logical reads  = 0
    Buffer pool temporary index physical reads = 0
```

```
Buffer pool xda logical reads              = 0
Buffer pool xda physical reads             = 0
Buffer pool temporary xda logical reads    = 0
Buffer pool temporary xda physical reads   = 0
Blocking cursor                            = NO

Dynamic SQL statement text:
select job,workdept,sex,count(*) from employee group by cube(sex,job,workdept)
.........................................略.........................................
```

1.1.7 收集进程/线程堆栈信息

在对 DB2 故障进行诊断的过程中，有些情况可以通过 DB2 直接返回的错误信息进行诊断处理，有些情况可以通过 db2diag.log 日志文件中的信息进行诊断处理。但在有些情况下，并不能通过上述方法或其他直观方法确定故障的原因及解决方法，此时可以使用 DB2 的 stack 和 trace 信息进一步深入地进行故障分析。

DB2 的 stack 信息中保存了目标 EDU 线程当前的 stack 信息，从这些信息中可以看到获得 stack 信息的瞬间，这个 EDU 线程的程序调用情况和执行状态。DB2 的 stack 信息一般有两个途径可以获得：当 DB2 中发生严重异常时会主动抛出一些 stack 信息，在这种情况下，直接读取分析这些信息即可；在另外的情况下，就只能通过 db2pd 命令主动获取 stack 信息，简化语法如下：

```
db2pd -stack [eduid/ pid/all]
```

DB2 的 trace 信息中保存了目标线程或所有线程在某一个时间段内程序的调用情况和执行状态。DB2 的 trace 信息可以通过 db2trc 或 db2pd 命令的特定选项获取到。db2trc 的简化语法如下：

```
db2trc on -p <pid.tid> -f <tid.trc>
30 秒或 10 秒或 5 秒
db2trc off
db2trc fmt -t <tid.trc> <tid.fmt>
db2trc flw -t <tid.trc> <tid.flw>
```

注意：
tid 指的是 kernel tid，获得此信息需要在获得 eduid 后使用 db2pd –edus 进行转换。

通过分析 stack 或 trace 信息可以知道故障时 DB2 的线程在执行哪些程序，从而判断出可能引起故障或性能问题的原因。

当然，由于我们没有 DB2 的源代码，因此无法获得某个程序、函数的具体执行内容，

但大多数情况下通过一些函数名称和执行顺序我们还是能够分析和判断出主要的故障原因，比如性能问题中最常见的 getConflict 等。所以，这些分析和判断主要还是基于对 DB2 产品的了解和经验。因此多多实践对于此项技能的提升十分必要和重要。

1.2　代理程序通信

1.2.1　代理程序概述

DB2 的代理(agent)是位于 DB2 服务器中服务于应用程序请求的一些进程或线程。当有外部应用程序连接至 DB2 实例并提出访问请求时，DB2 的代理就会被激活去应答这些请求。一般 DB2 的代理被称为工作代理，工作代理大概有三种类型：空闲代理、活动的协调代理、子代理。

- 空闲代理：指的是没有任何任务的代理。这种代理既不服务于任何远程连接，也不服务于本地连接，处于一种备用或待命状态。
- 活动的协调代理：指的是处于工作状态的代理，每个外部应用程序产生的数据库活动连接都有活动的协调代理来为之服务。
- 子代理：指的是接受协调代理分发出来的工作的下一级代理。在 DB2 V9.5 以前，只有在多分区环境(MPP)或节点内并行(INTRA_PARALLEL=ON)环境下才存在子代理，从 DB2 V9.5 开始所有环境中都可能存在子代理。

在 DB2 服务器中存在代理池，当实例刚启动后这里便有一些代理(数量取决于实例参数 num_initagents)。在没有任何数据库连接时，它们处于待命状态，就是空闲代理。而当有外部程序连接至数据库时，这些代理开始得到命令去服务于这些新建的连接，这时它们就变成了活动的协调代理。这些协调代理再将请求逐步细分，分配给下一级代理，即子代理去处理。如果当前的代理都已经在工作了，同时又来了新的请求，数据库管理器会产生新的代理去应答。当事务处理完毕而且数据库连接断开后，协调代理要么返回代理池，变回空闲代理，要么就自动销毁(取决于实例参数 num_poolagents)。这就是代理的生命周期。

1.2.2　代理程序相关配置参数

通过执行 db2 get dbm cfg 可以看到以下几个和代理相关的实例参数：

- **NUM_POOLAGENTS**：这个参数用来控制代理池中空闲代理的数量。当活动的代理完成工作返回代理池变成空闲代理时，如果数量超过这个参数，那么这个代理就会自动销毁。注意：在连接集中器激活的情况下，代理池中的空闲代理数目在某一时刻可能会超过 NUM_POOLAGENTS 的大小以应对突发的高密度连接。

- **NUM_INITAGENTS**：这个参数就是前面提到的在实例刚刚启动时便生成的一些空闲代理的数目。这是为了提高性能，因为这些代理可以随时变成协调代理去应答外部应用请求，而不用临时再生成新的代理。
- **MAX_COORDAGENTS**：这个参数决定了在实例中，在同一时刻最大的协调代理的数目(在多分区环境中指的是节点上的最大协调代理数)。
- **MAX_CONNECTIONS**：这个参数决定了允许连接至某个实例的最大连接数(在多分区环境中指的是节点上的最大连接数)。

还有一个和连接相关的 DB2 之配置参数 **MAXAPPLS**，可以通过“db2 get db cfg for database_name”得到，这个参数决定了同时连接至某个数据库的最大连接数。处于实例之下的所有数据库的 MAXAPPLS 值之和不能超过实例参数 MAX_CONNECTIONS。

针对上面几个配置参数，下面我们举一个例子，如下所示：

```
Max number of existing agents         (MAXAGENTS) = 200
Agent pool size                       (NUM_POOLAGENTS) = 100(calculated)
Initial number of agents in pool      (NUM_INITAGENTS) = 0
```

由于 NUM_INITAGENTS 是 0，因此在运行 db2start 时不会显示“db2agent(idle)”进程。例如，如果在运行 db2start 之前将 NUM_INITAGENTS 设置为 5，那么在发出 db2start 之后，将显示下面这些额外进程：

```
db2inst1 35542 59814   0 16:25:57      -  0:00 db2agent (idle)
db2inst1 43096 59814   0 16:25:57      -  0:00 db2agent (idle)
db2inst1 49628 59814   0 16:25:57      -  0:00 db2agent (idle)
db2inst1 58170 59814   0 16:25:57      -  0:00 db2agent (idle)
db2inst1 64012 59814   0 16:25:57      -  0:00 db2agent (idle)
```

当应用连接到 SAMPLE 数据库后，“db2agent (SAMPLE)”进程出现了。该进程表明实际上存在到 SAMPLE 数据库的连接。如果我们发出以下命令：

```
db2 connect reset
```

“db2agent(SAMPLE)”现在将变成“db2agent(idle)”。这是因为 NUM_POOLAGENTS 被设置成大于零的数，这意味着该代理程序尽管是空闲的，但在代理池中将仍然保持被分配的状态。如果 NUM_POOLAGENTS 被设置成零，那么在运行了“connect reset”后，将不再有“db2agent”进程运行。

在数据库和相应的数据库实例中，必须存在空闲或可用的代理程序，才能处理发送到数据库的请求，所以需要对数据库和实例中的代理程序情况进行监控，以及时发现代理程序存在的问题。

在使用 db2start 启动实例的时候，会根据参数 NUM_INITAGENTS 的值分配空闲的代理程序，在应用连接到数据库请求数据的时候，这些代理程序都会被分配给应用来执行对数据的访问。当初始化创建的空闲代理程序都已经被使用之后，就需要创建新的代理程序来满足需求。但是在实例内创建的代理程序的总数量不能超过实例参数 MAXAGENTS 的设置。当代理程序总的数量达到这一限制之后，就只能从其他空闲的应用中偷出(stolen)代理程序。而随着数据库负载的降低，需要的代理程序数量也随着降低，但是不需要的代理程序不会被立刻销毁，而是变成空闲的代理程序。但是当空闲的代理程序数量达到 NUM_POOLAGENTS 参数的设置之后，空闲下来的代理程序将不会被保留，而是被销毁。

当应用连接到数据库的时候，数据库需要向应用分配代理程序以执行对数据的访问。数据库在从代理程序池中获得空闲的代理程序之后，并不能马上将其分配给应用，而是要为代理程序获得令牌(token)，在成功获得了令牌后才能将其分配给应用。那么数据库实例中的令牌数量也就是当前正在执行的代理程序的数量，因而令牌的最大值就控制了实例中可以并发执行的代理程序总数，此值由实例参数 MAXCAGENTS 设置。代理的工作原理如图 1-5 所示。

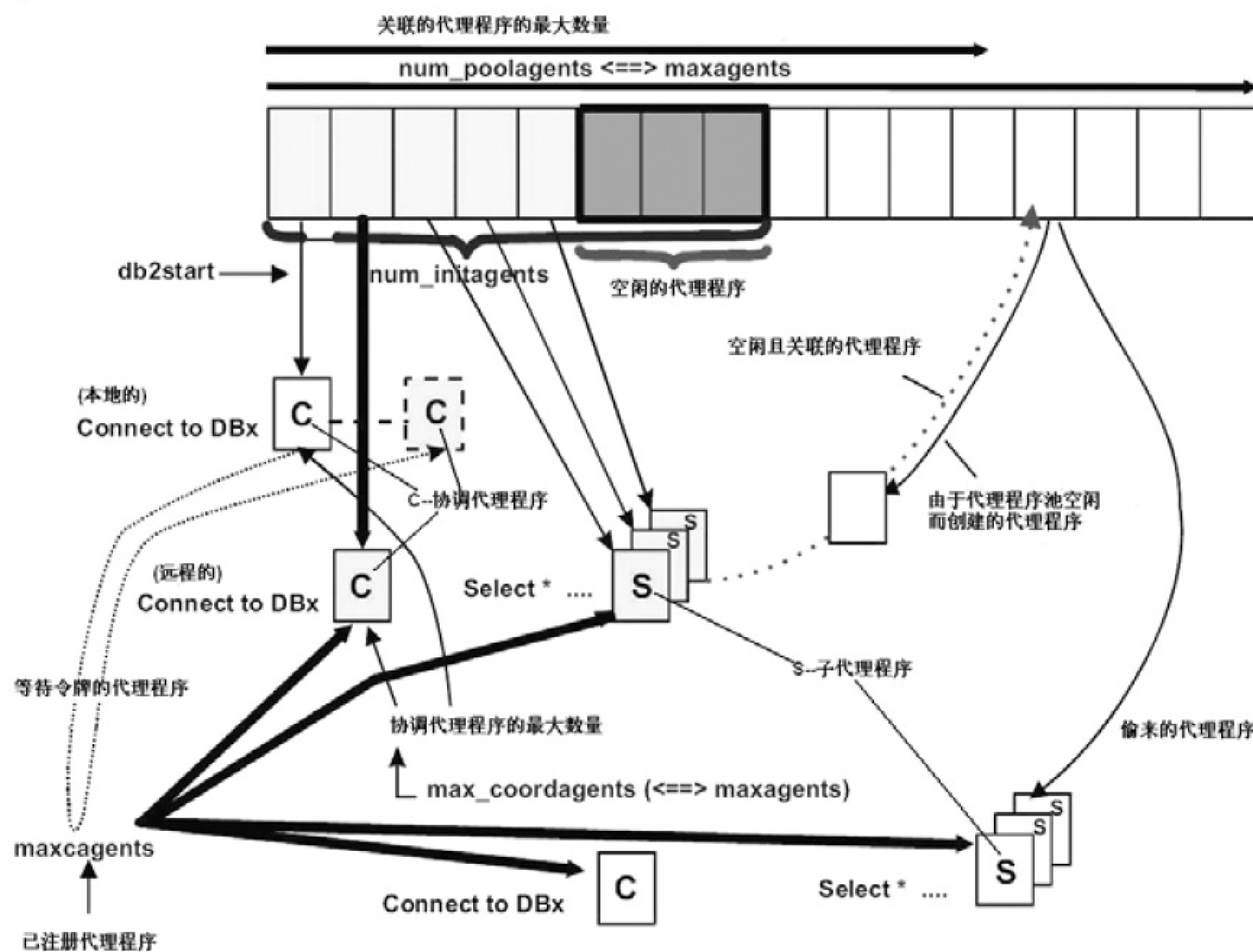


图 1-5　代理的工作原理

通过实例快照和数据库快照可以分别观察到实例级别和数据库级别的代理程序的使

用情况，根据实际情况与相关的参数做比较，评估是否需要修改这些参数来满足代理程序的使用需求。

1.2.3 应用程序、代理程序和事务

下面我们简单地解释一下应用程序、代理程序(代理进程)和事务之间的关系。可能很多刚刚接触 DB2 的读者经常被那些诸如应用程序、代理程序、事务、连接集中器(CONCENTRATOR)、INTRA_PARALLEL 等之类的概念弄得头晕脑胀，现在我们用一个银行取钱的例子解释一下这些概念之间的关系。

首先我们想象有一家银行，叫作 EBANK，这家银行就相当于我们生活周围的某个银行。某一天，有个人跑到 EBANK 银行，这个人叫小牛。小牛就是所谓的应用程序(APPLICATION)，他跑过来到银行办理业务。小牛要干嘛？不存钱不取钱，只是查账，也就是 query only 类型的应用程序。小牛跑进了银行大厅，首先拿号，这个号就叫作应用程序 ID(应用程序句柄，APPLICATION HANDLE)，每一个跑到银行办业务的人都需要排号。如果下次来，还要拿号，号码可能不一样……。

现在我们来看看什么是连接集中器。所谓的连接集中器就是规定，派发出去的号码是否能够超过当前柜员的数量，这里的柜员就是代理程序(agent)，他们为顾客服务。如果银行规定派发出去的号码的数量不能够超过柜员的数量，也就是说没有打开连接集中器，那就可以确保每个顾客都有至少一个柜员为之服务。但是如果银行说号码随便发，多来的人排队等着(好像大部分银行都那样)，这也就是打开了连接集中器，进来的连接数目可以超过代理进程的数量。这个时候小牛同学运气不错，很多窗口都是空的，于是小牛来到一个窗口前，柜员热情地接待了小牛，柜员问：1234 号柜员为您服务，请问办理什么业务并请先输入您的密码。这里的 1234 号就是 DB2 代理进程 ID。小牛输入自己的账号密码，这就是连接的认证(AUTHENTICATION)部分。看到密码 OK 了，柜员问：先生需要什么服务？小牛说：俺要查账，看看俺还有多少钱。这个自然就是应用程序发起的查询。然后，柜员就开始为小牛服务了，动动鼠标打开电脑，输入自己的账号密码，这就是所谓的事务(TRANSACTION)的开始，银行里对所有操作都有记录，这个记录的号码就是事务 ID……，在事务期间，应用程序的状态为 UOW EXECUTING 状态。这时候小牛拿出一个很长很长的单子，说俺的账号比较多(小牛比较有钱)，你们最好帮忙查一下这张单子上所有账号都有多少钱……看着密密麻麻的银行账号，柜员脸儿都绿了。此时，如果这家银行允许柜员之间互相帮忙(INTRA_PARALLEL=ON，激活分区内并行)，这个柜员就可以叫另外一个或多个空闲的银行后台工作人员帮忙查询这些单子上的账号，此时为小牛同学服务的代理叫协调代理(COORDINATOR AGENT)。但是如果没有激活分区内并行，这个可怜的代理进程就只能自己慢慢做了。所以启用分区内并行可以提高查询(select)的速度，对一些执行时间比较长的

比较复杂的 SQL 语句尤其有效，因为可以利用多个代理进程来完成任务。这一笔好长好长的单子终于完了，柜员松了口气，小牛还坐在柜台前看自己长长的单子，死活不肯离去……，在此期间，小牛这个应用程序还占着窗口不走(此时应用程序处于状态 UOW WAITING)，也不办理业务(如果办理业务，在 DB2 里面该应用程序就状态 UOW EXECUTING)，搞得后面好多客户都没有窗口只能排队(窗口数量就是 MAX_CONNECTIONS 的设置，柜员的数量就是 MAXAGENT，如果窗口被占满，后面的那些应用程序就连不上数据库)。在启用连接集中器的情况下，也就是说，在柜员完成一单交易后，不论客户走了没有，只要他当时没有其他请求，柜员就要服务其他的客户……。

1.2.4　代理和连接的常见问题与优化

代理程序和连接的常见问题主要有以下几类：

1. 连接超限问题

在 DB2 中，当设置的 MAX_CONNECTIONS 值比较小时，如果出现外部连接数过多，就会出现错误。db2diag.log 诊断日志的输出如下所示：

```
PID     : 762076             TID  : 772        PROC : db2acd
INSTANCE: db2inst1            NODE : 000
APPID   : *LOCAL.db2inst1.080115203015
EDUID   : 772                EDUNAME: db2acd
FUNCTION: DB2 UDB, DRDA Communication Manager, sqljcReceive, probe:30
MESSAGE : ZRC=0x8136001C=-2127167460=SQLZ RC NO CONNECTION, SQLT SQLJC
          "No connection"
DATA #1 : String, 11 bytes
CCI Error:
DATA #2 : unsigned integer, 8 bytes
...
```

这时可以通过下面的命令来查看当前的连接数：

```
    $ db2 list applications Auth Id  Application    Appl.   Application Id
DB  # of       Name         Handle                        Name    Agents
    -------------------------------- ----------------- -------- -----
    DB2INST1 db2taskd     583      *LOCAL.db2inst1.080112150958    SAMPLE    1
    DB2INST1 db2stmm      582      *LOCAL.db2inst1.080112150957    SAMPLE    1
    DB2INST1 java         592      *LOCAL.db2inst1.080115201505    SAMPLE    1
    DB2INST1 java         572      *LOCAL.db2inst1.080115201445    SAMPLE    1
    DB2INST1 java         585      *LOCAL.db2inst1.080115201458    SAMPLE    1
    DB2INST1 java         565      *LOCAL.db2inst1.080115201437    SAMPLE    1
```

```
    DB2INST1 java        584       *LOCAL.db2inst1.080115201457   SAMPLE   1
    DB2INST1 java        590       *LOCAL.db2inst1.080115201503   SAMPLE   1
    DB2INST1 db2bp       591       *LOCAL.db2inst1.080115201502
    ...
```

可以查看这时的连接数，并与 MAX_CONNECTIONS 的值进行比较，从而做出调整。这时应当注意，有两个服务器内部的特殊进程“db2stmm”和“db2taskd”不应算作外部连接。“db2stmm”是用来管理内存自动调优的代理，“db2taskd”是用来分配数据库后台任务的代理。示例中的“java”代表外部连接来自 Java 应用程序。“db2bp”代表来自 DB2 CLP 的连接。可以看到这些连接都连到了数据库 SAMPLE 上。

接下来可以通过 db2pd 命令查看当前的代理数，输出结果如下所示：

```
     $ db2pd -agents -db SAMPLE
    Option -agents is an instance scope option.  The database option has been
ignored.
    Database Partition 0 -- Active -- Up 5 days 05:32:31
    Agents:
    Current agents:      9
    Idle agents:        0
    Active coord agents: 6
    Active agents total: 6
    Pooled coord agents: 3
    Pooled agents total: 3
    Address    AppHandl [nod-index] AgentEDUID Priority  Type    State
ClientPid  Userid   ClientNm Rowsread  Rowswrtn   LkTmOt DBName
    0x7AB98C80 156      [000-00156] 3560       0         Coord    Inst-Active
5036       ORACLE   db2bp.ex 249        0         NotSet SAMPLE
    0x7AB99E70 157      [000-00157] 3320       0         Coord    Inst-Active
5036       ORACLE   db2stmm  0          0         NotSet SAMPLE
    0x7AB9B2A0 158      [000-00158] 536        0         Coord    Inst-Active
5036       ORACLE   db2taskd 2          0         NotSet SAMPLE
    0x7AB9C6D0 159      [000-00159] 5468       0         Coord    Inst-Active
5036       ORACLE   db2wlmd  0          0         NotSet SAMPLE
    .................略..........................
```

在这里可以看到，“Idle agents”的值为 0，这表明代理池中已经没有空闲代理了(进程的“State”全都是“Inst-Active”)。这时可以将“Current agents”的值与 MAXAGENTS 的值做比较，或者将“Active agents total”的值与 MAX_COORDAGENTS 的值做比较，从而做出相应调整。

对于这种问题，还可以通过分析数据库管理器的快照来做出调整，快照输出如下所示：

```
db2 get snapshot for dbm:
...
Remote Connection Executing in the Database Manager = 58
Local Connection Executing in the Database Manager  = 1
...
Agents assigned from pool                         = 38
Agents created from empty pool                     = 158
Agents stolen from another application             = 1
High water mark for coordinating agents            = 60
Max agents overflow                              = 3
Hash joins after heap threshold exceeded           = 0
......
```

可以看到“Max agents overflow”的值等于 3，说明有 3 次生成代理数超过限制的情况。这时会在 db2diag.log 中看到前面的错误信息。此时必须调节 MAXAGENTS 的值以修复当前错误。可以将 MAX_COORDAGENTS 设定为与“High water mark for coordinating agents”相同的值，在单分区环境下可以将 MAXAGENTS 设定与 MAX_COORDAGENTS 一样，在多分区环境(DPF)或节点内并行环境(INTRA_PARALLEL=ON)中，根据节点数计算出结果 MAXAGENTS=(N+1)*MAX_COORDAGENTS(N 为分区数)。另一方面，在 MAX_COORDAGENTS 不是 automatic 的情况下，如果“Remote Connection Executing in the Database Manager”的值与“Local Connection Executing in the Database Manager”的值之和接近 MAX_COORDAGENTS，这时要适当增大 MAX_COORDAGENTS 的值。

一般说来有这样的原则，当在连接数据库出现内存错误时，调节如下参数：

- 在单分区并且没有节点内并行性的情况下增大 MAXAGENTS 的值。
- 在多分区(DPF)或节点内并行环境(INTRA_PARALLEL=ON)中增大 MAXAGENTS 或 MAX_COORDAGENTS 的值。
- 在连接集中器被激活的情况下，增大 MAX_CONNECTIONS 的值。

2. 连接挂起问题

还有一个与连接相关的问题是在首次连接数据库时，连接时间总要长一些。这是因为数据库在为首次连接分配内存，主要是缓冲池。连接时间的长短取决于操作系统的内存调用情况以及缓冲池的大小。有时用户常常会为了提高应用性能而盲目地扩大缓冲池，造成缓冲池设置得太大，甚至超过了数据库共享内存，使得实例无法为数据库分配足够的内存，在连接数据库时就会出现挂起现象。而这时想将缓冲池设小也没办法了，因为数据库连不上，无法设置缓冲池。这也是一个常见的问题。遇到这种问题时，有些用户甚至被迫重建数据库。其实这个问题可以通过设置 DB2 注册参数 DB2_OVERRIDE_BPF 来设置缓冲池

的大小，从而能够再次连接数据库。在默认情况下，缓冲池的大小被设置成－2(通过“select npages from syscat.BUFFERPOOLS”得到)，这说明缓冲池是自动增长的，这种情况下最好不要修改缓冲池的大小，可以让 DB2 自动去调节。这种问题的另外一种解决方法是修改 DATABASE_MEMEORY 参数，把这个参数从 automatic 或设置好的比较大的值修改为较小的值，保证数据库能够连接上，然后再修改缓冲池的大小。

3. 常见通信错误

通常在连接数据库时还会遇到一些与网络通信相关的错误，这些错误代码有 SQL30080、SQL30081N 等。可以用以下一些方法来尝试解决：

- 执行命令“db2set –all”，检查一下是否有 DB2COMM=TCPIP 一项。如果没有，那么应该执行“db2set DB2COMM=TCPIP”来进行设置。
- 执行命令“db2 get dbm cfg | grep SVCENAME”，检查 SVCENAME 设定的服务是否已在“/etc/services(UNIX)”中定义(在 Windows 环境中是在“%windir%\system32\drivers\etc\services”)。当然，如果 SVCENAME 是端口号(端口号应小于 65536)，那么不用在“/etc/services”中定义。
- 执行命令“netstat –a”，检查输出中是否有“/etc/services”中定义的端口或服务在监听(listen)。如果没有，那么可能需要重启网络或机器。
- 这种问题也可能是由防火墙导致的，在 Linux 上可以通过编辑“/etc/sysconfig/iptables”文件来绕过防火墙(这需要 root 权限)。

如果遇到其他特殊的问题，可以通过命令“DB2 ? SQLxxxxx”根据得到的提示去分析具体问题。

4. 性能优化

- 调节 NUM_POOLAGENTS

对于决策支持系统，由于连接数较少，NUM_POOLAGENTS 可以设为较小的值，从而避免过多的空闲代理而浪费资源。而对于在线事务处理系统，由于连接数较多，可以设为较大的值，从而减少因为频繁创建和删除代理而产生的系统消耗。具体数值可以通过分析数据库管理器快照来进行调节，通过分析数据库管理器快照来调节 NUM_POOLAGENTS 的输出结果如下所示：

```
db2 get snapshot for dbm
...
Agents assigned from pool                    = 38
Agents created from empty pool                 = 158
```

```
    Agents stolen from another application              = 1
    ...
```

当“Agents created from empty pool/Agents Assigned From Pool”的比值较小时，说明代理的重用率比较高。当比值比较大时，说明这时代理的创建、删除比较频繁，此时需要增大“NUM_POOLAGENTS”来减少系统频繁创建、删除代理时的资源消耗。当“Agents stolen from another application”的值较大时，也应当增大 NUM_POOLAGENTS 的值。当然如果 NUM_POOLAGENTS 设置太大，可能会产生很多不必要的空闲代理长时间滞留在代理池中，造成资源的浪费。在 DB2 V9.5 及以上版本中，NUM_POOLAGENTS 的默认值被设为 automatic(初始值为 100)，这样数据库管理器便可以自动管理代理池中空闲代理的数目。

- 调节 NUM_INITAGENTS

NUM_INITAGENTS 的值最好和 NUM_POOLAGENTS 的值一致。这样可以减少处理事务时生成代理的时间，而将这部分等待时间转移到启动实例时，这对用户来说是最理想的。

- 调节 MAX_CONNECTIONS 与 MAX_COORDAGENTS

激活连接集中器，即设定 MAX_CONNECTIONS 大于 MAX_COORDAGENTS，这样可以节省 DB2 代理的数目，减少资源消耗，扩大连接数。在 DB2 V9.5 及以上版本中，最好将 MAX_CONNECTIONS 与 MAX_COORDAGENTS 都设为 automatic，这样可以让 DB2 自动根据连接数调节代理数。

1.3　实用程序相关进程

DB2 中提供了一些用来移动和维护数据的实用程序，这些实用程序可以帮助 DBA 装载、备份、恢复数据库。为满足大数据量数据处理时的高性能要求，实用程序也有自己相关的进程。

1.3.1　LOAD 相关进程

我们知道 LOAD 是 DB2 数据库中使用非常广泛的快速数据移动工具，LOAD 本身也比较复杂，在做 LOAD 时数据库也会产生很多进程，DB2 LOAD 的进程体系结构如图 1-6 所示。

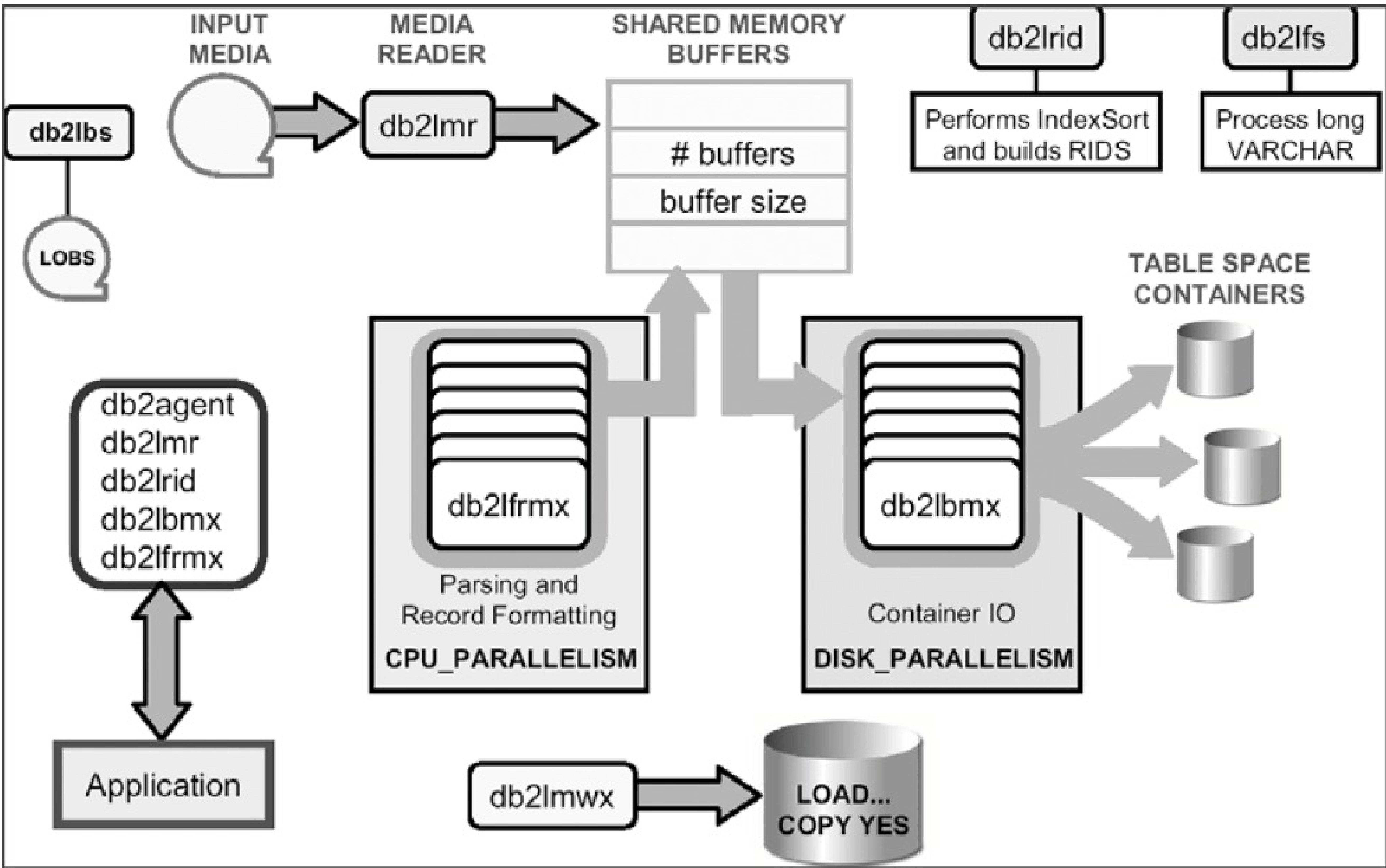


图 1-6　LOAD 相关的进程体系结构图

LOAD 相关的进程和进程描述如表 1-2 所示。

表 1-2　DB2 LOAD 相关的主要进程及描述

进　程　名	描　　述
db2lbs	LOAD LOB 扫描程序。仅当 LOAD 工具正在装入带有 LOB 列的表时才使用它们。这些进程扫描表的 LOB 对象，并将该信息读回表中
db2lbmX	LOAD 缓冲区操纵器。最后那个字符“X”表示 1 或更大的数字。该进程将已装入的数据写到数据库，并且可能涉及异步 I/O。“X”始终是 1，不过通常也会是更大的数字，这取决于试探值(heuristic)。试探值取决于系统上的 CPU 数以及被写的容器数 这个“智能的默认值”可能会被 LOAD 命令的 DISK_PARALLELISM 修饰符覆盖 我们应当明白，这个异步 I/O 不是某些操作系统支持的异步文件 I/O；而只意味着我们有一些写 I/O 的独立进程。这意味着正在格式化数据的其他进程不用被 I/O 等待所束缚
db2lfrmX	LOAD 格式化程序进程。最后那个字符“X”表示 1 或更大的数字。该进程将输入数据格式化成内部格式。它始终出现在 LOAD 中。该进程使用智能的默认值，可能会被 CPU_PARALLELISM 修饰符覆盖以选择最佳的 CPU 数
db2lfs	当被装入的表包含 LONG VARCHAR 列时使用这些进程，这些进程用来读和格式化表中的 LONG VARCHAR 列

(续表)

进　程　名	描　　述
db2lmr	这是 LOAD 媒体阅读器(Media Reader)进程。它读取装入的输入文件，一旦读完所有输入文件，该进程就会消失。甚至在整个装入操作完成之前，该进程就会消失
db2lmwX	这些是 LOAD 媒体记录器进程。最后那个字符“X”表示 1 或更大的数字 如果为 LOAD 命令指定“装入副本”(load copy)选项，那么该进程将生成装入副本。装入副本本质上就是装入到表中的数据备份 这些媒体记录器与 BACKUP 和 RESTORE 使用的媒体记录器相同。就像在命令行上描述的那样，每个复制会话调用一个媒体记录器(可以创建多个文件的装入副本)。如果没有装入副本，就没有媒体记录器。它们根据数据的类型在装入时从其他进程获取输入，但是由缓冲区操纵器写的每位数据通常都将被传递到媒体记录器。就如同其他所有进程那样，它们由装入代理程序控制
db2lrid	该进程在 LOAD 期间执行索引排序并构建索引记录标识(Record ID，RID) 该进程不会出现在非并行数据库实例(禁用 INTRA_PARALLEL 的实例)中。该进程执行的任务由非并行实例中的格式化程序 EDU 完成 该进程完成下列三种功能： ● SMP 同步 ● 分配 RID，最后那个负责构建索引 ● 控制 LOAD 格式化程序进程的同步
db2ltsc	LOAD 表扫描程序。这些进程扫描数据对象、查找被装入的表并读取 LOAD 工具的信息 在 LOAD 追加操作过程中使用这些进程

在分区数据库环境下，与 DB2 LOAD 相关的进程如图 1-7 所示。

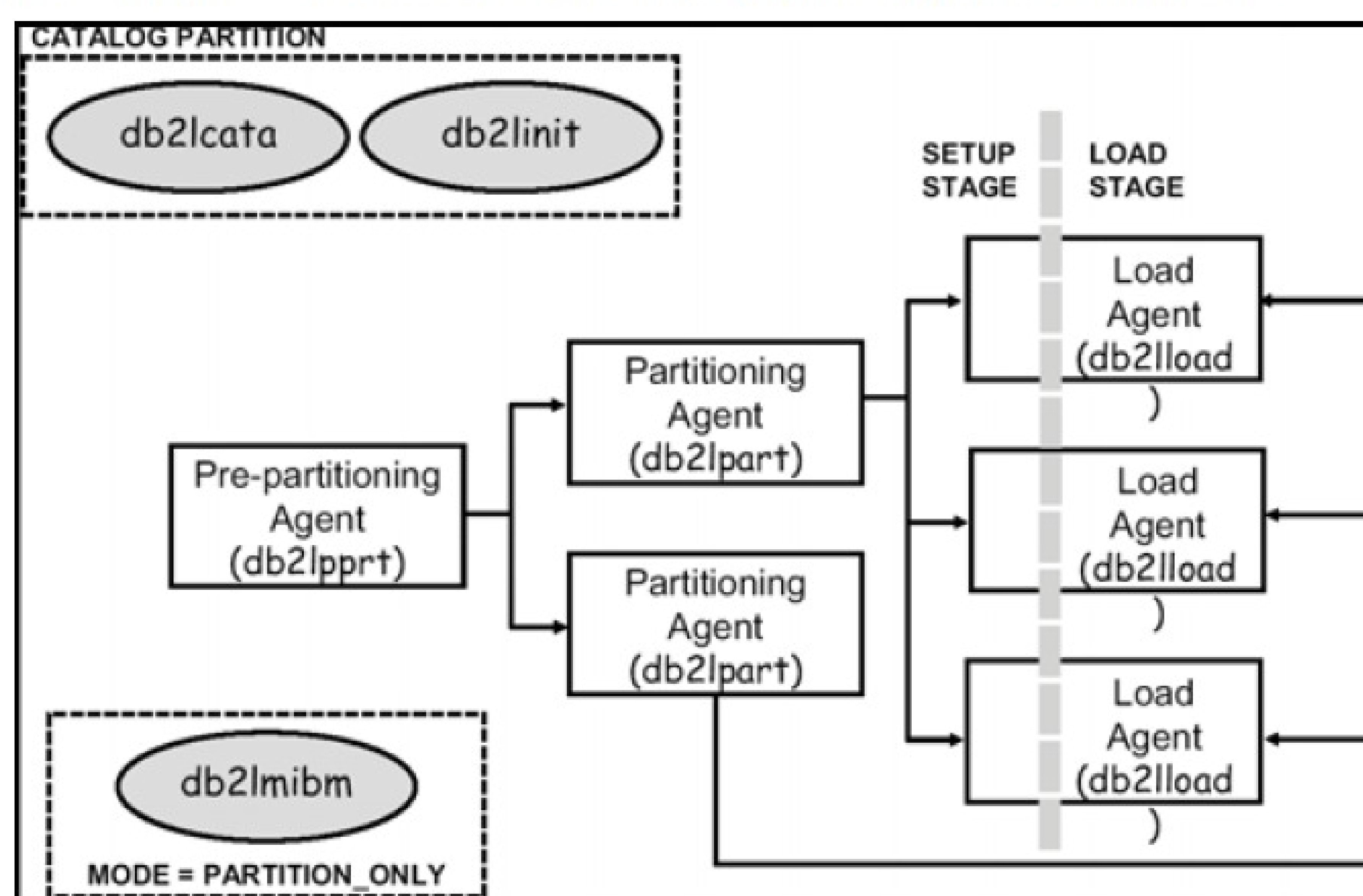


图 1-7　分区环境下的 DB2 LOAD 进程体系结构

分区环境下和 LOAD 相关的进程和进程描述如表 1-3 所示。

表 1-3　分区环境下的 DB2 LOAD 常见进程及描述

进　程　名	进 程 描 述
db2linit	LOAD 初始化子代理程序。这个子代理程序获取数据库分区上必需的资源，并将应答序列化，返回给装入目录子代理程序
db2lcata	LOAD 目录子代理程序。这个子代理程序只在目录分区上执行，负责： ● 衍生初始化子代理程序 ● 处理应答 ● 存储目录分区上的锁信息 LOAD 目录子代理程序还查询系统目录表以确定哪些分区用于数据分割和分区 正常的装入作业只有一个目录子代理程序。异常情况是装入无法获取某些分区上的装入资源如果数据库分区上的设置错误被隔离出来，那么协调程序将从装入的内部分区列表除去发生故障的分区，并衍生新的目录子代理程序。这一过程会重复进行，直到成功获取所有分区上的资源，或者在所有分区上都遇到了故障
db2lpprt	装入预分区子代理程序。这个子代理程序将输入数据从一个输入流预分区成多个输出流，每个分区子代理程序都有一个这样的进程 每个输入流都将有一个预分区子代理程序
db2lpart	装入分区子代理程序。这个子代理程序将输入数据分区成多个输出流，将写入数据的每个数据库分区都有一个这样的进程 分区子代理程序的数量可以由用户进行配置，默认数量取决于输出数据库分区的总数
db2lmibm	装入微型缓冲区操纵器子代理程序如果为装入使用了 partition_only 方式，那么该子代理程序就编写分区的输出文件 每个输出数据库分区就将有一个微型缓冲区操纵器子代理程序
db2lload	装入子代理程序。这个子代理程序负责完成每个数据库分区上的装入操作：衍生格式化程序、ridder、缓冲区操纵器和媒体记录器 EDU，并监视它们的工作 每个输出数据库分区都将有一个装入子代理程序
db2lrdfl	装入读文件子代理程序。这个子代理程序读取给定数据库分区上的消息文件，并将数据发送回客户机。每个输出分区、分区的分区和预分区的分区都将有一个装入读文件子代理程序
db2llqcl	装入查询清除子代理程序。这个子代理程序从给定分区除去所有装入的临时文件 每个输出分区、分区的分区和预分区的分区都将有一个装入查询清除子代理程序
db2lmitk	装入微型任务子代理程序。这个子代理程序释放了在某次从游标调用的装入或 CLI 装入中使用的所有 LOB 定位器 运行在协调程序分区上的每个游标/CLI 装入都将有一个装入微型任务子代理程序

(续表)

进　程　名	进 程 描 述
db2lurex	装入用户出口子代理程序。这个子代理程序运行用户的文件传送命令，使用文件传送命令选项的每个装入作业都将有一个装入用户出口子代理程序
db2lmctk	这个子代理程序用于持有、释放或降级(downgrade)目录分区上持有的由于装入而产生的锁

LOAD 工具与其他数据移动工具比较起来的一大优势就是提供卓越的性能，这主要是由于 LOAD 在对数据加载时采取数据页级别的处理，这绕过了数据库管理系统的多个处理层次，因此可以极大地提高性能。除了 LOAD 工具本身的这一特点之外，我们还可以通过合理设置 LOAD 的一些选项来进一步提高其性能。下面列出一些影响 LOAD 性能的选项及其合理设置的建议。

CPU_PARALLELISM n

此选项用于指定 LOAD 同时使用 n 个 CPU 来并发处理 LOAD，在 LOAD 处理的数据量较大并且操作系统的负载不高的情况下，可以通过此参数指定多个 CPU 并发地执行表构建过程中的解析、转换、格式化等内容来提高效率。如果同时启动了多个 LOAD 工具，就要注意，为所有 LOAD 工具指定的此参数最好不要超过操作系统中逻辑 CPU 的总数(在此指定的 CPU 是 LCPU，即逻辑 CPU)。

此选项若不设置，DB2 会根据当前操作系统中 CPU 的数量自动分配 CPU。

DATA BUFFER buffersize

此选项用于指定 LOAD 工具能够使用的数据缓存的最大值，单位是 4KB。我们可以想象，在处理的数据量很大，并且在不超过操作系统空闲物理内存的情况下，我们为 LOAD 分配越多的数据缓存，LOAD 的性能将会越好。但是此值的设置受到数据库参数 util_heap_sz 的限制。由于使用 util_heap_sz 的工具有多个，因此为某个 LOAD 分配的数据缓存一般建议不要超过 util_heap_sz 的 50%。在实际设置时，最好根据 util_heap_sz 的实际情况来确定，这可以通过观察数据库快照中工具堆使用的大小和高水位大小来判断。

另外，此选项并非单纯的设置越大越好，因为在数据缓存设置得足够大以后，即使再增加其大小也不会有利于性能的提升，因为性能的瓶颈已经不再是缓存了。所以需要在实际生产中做多次测试才能找到最适合的值。

DISK_PARALLELISM n

此选项用于指定 LOAD 工具可以利用向表空间中多个容器执行并发 I/O 的能力来提高

性能。根据表空间中容器的数量做适当设置即可。

ANYORDER

此文件修饰符可以使用输入文件中预设好的排序结果来提高性能。如果输入文件来自于 EXPORT 工具中使用一定排序谓词导出的数据，那么性能会得到较大提升(可以提升几倍到几十倍)。此文件修饰符可以用于各种输入文件格式。

FASTPARSE

此文件修饰符通过降低对输入数据的检查来提升性能，如果输入数据与目标表在结构、类型上并无差别，那么可以通过设置此选项来提升性能。此文件修饰符只能用于 ASC 或 DEL 格式的文件。

NOROWWARNINGS

如果预期在 LOAD 过程中会有大量的警告出现，那么可以使用此文件修饰符来提升性能。

SAVECOUNT

此选项可以降低 LOAD 在设置一致性恢复点上的负载，在处理大数据量的情况下可以使用此选项来提升性能，根据处理数据量的情况来合理设置此值。如果需要加载的数据为 1000 万行，那么可以将此值设置为 10000。此文件修饰符不能与 ANRORDER 文件修饰符同时使用。

USER tablespace

当需要 LOAD 的表中存在大量索引需要构建时，并且在表或索引本身所在的表空间不是很大的情况下，使用此选项制定系统临时表空间可以提高数据构建的性能。

1.3.2　备份/恢复相关进程

在数据库中进行备份/恢复是我们经常做的操作，在备份/恢复时也会产生一些进程，了解这些进程对我们诊断数据库在备份/恢复时出现的问题会很有帮助，备份的进程模型如图 1-8 所示。

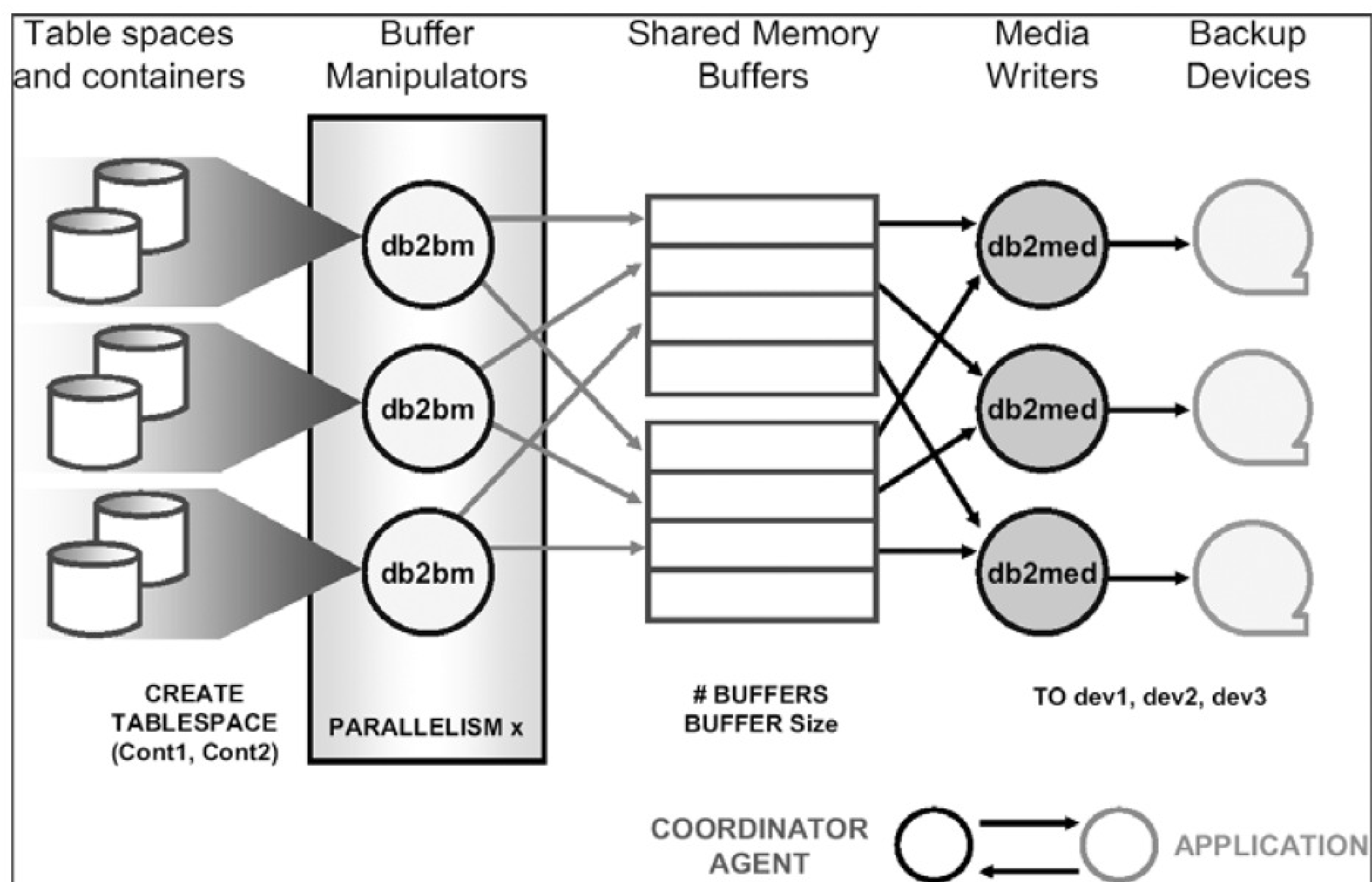


图 1-8　备份的进程模型

和恢复相关的进程如图 1-9 所示。

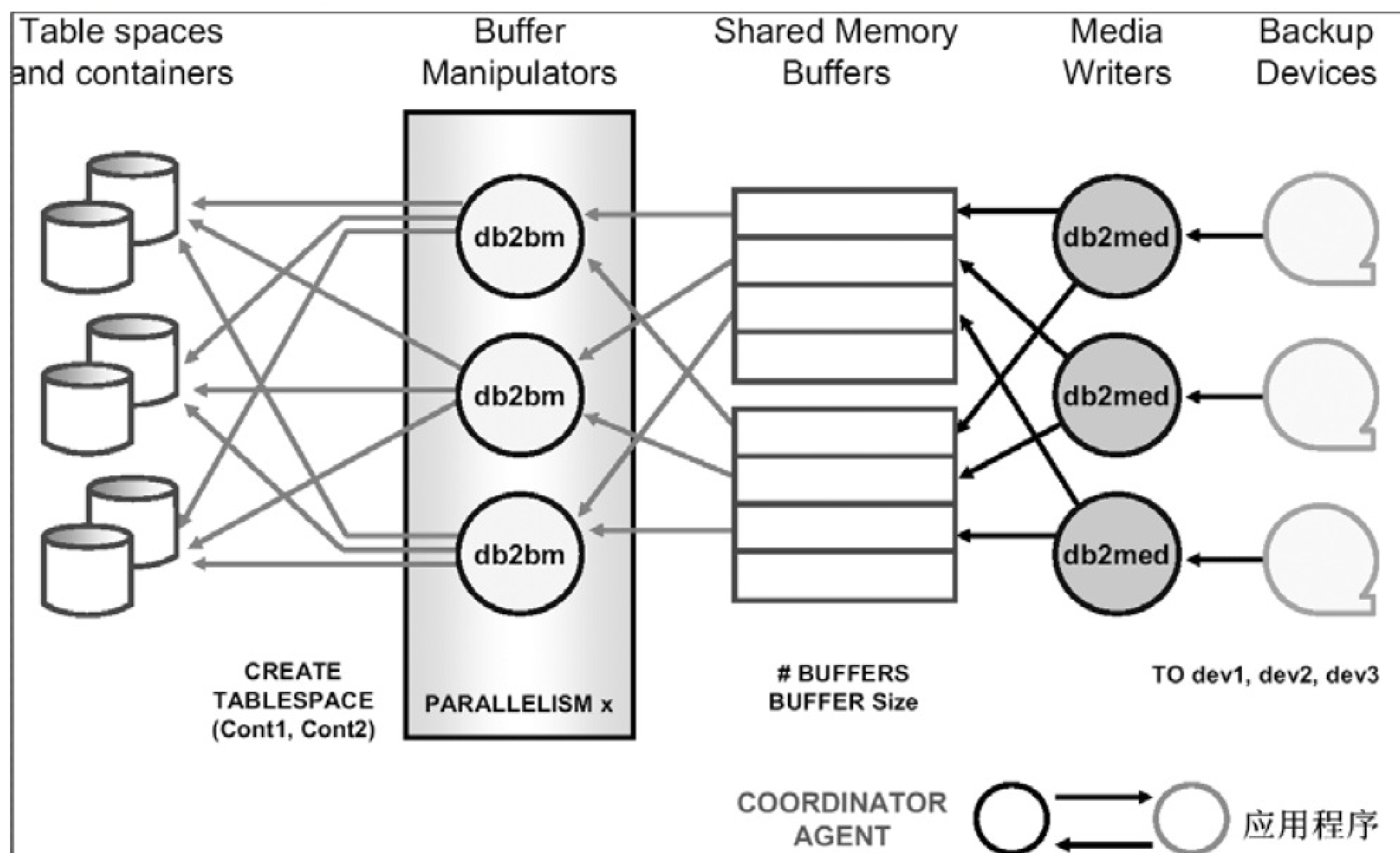


图 1-9　恢复相关进程

备份和恢复相关的进程及进程描述如表 1-4 所示。

表 1-4　DB2 备份与恢复时的常见进程

进　程　名	进 程 描 述
db2bm	备份/恢复缓冲区操纵器。该进程用于在备份操作过程中从表空间进行读取数据，以及用于在恢复操作过程中将数据写到表空间。通过 BACKUP 或 RESTORE 命令配置的每个备份/恢复缓冲区都将有一个 db2bm 进程。在 DB2 V9 以后版本中，可以按如下所示标识备份和复原线程： db2bm.%1.%2 是备份和复原缓冲区操纵程序，而 db2med.%1.%2 是备份和复原介质控制器，其中： %1：控制备份或复原会话的代理程序的 EDU 标识。 %2：用于区分属于特定备份或复原会话的线程(可能有许多个)的顺序值。 例如，db2bm.13579.2 可标识具有 EDU 标识为 13579 的由 db2agent 线程控制的第 2 个 db2bm 线程
db2med	这些进程对用于 LOAD、备份和恢复的数据库表空间进行读和/或写操作。它们将已格式化页面中的数据写到表空间容器

执行备份操作时，DB2 将自动为缓冲区个数、缓冲区大小和并行性设置选择最佳值。这些值根据可用实用程序堆内存的数量、可用处理器数和数据库配置而定。因此，根据系统中可用的存储量，应考虑通过增大 UTIL_HEAP_SZ 配置参数来分配更多内存。目的是最大程度上减少完成备份操作所需的时间。除非显式地输入以下 BACKUP DATABASE 命令参数的值，否则 DB2 将为它们选择一个值：

- WITH num-buffers BUFFERS
- PARALLELISM n
- BUFFER buffer-size

如果因为未指定缓冲区数和缓冲区大小而导致 DB2 设置这些值，那么对大型数据库的影响应该最低。但是，对于小型数据库来说，会导致备份映像大幅增大。即使写入磁盘的最后一个数据缓冲区只包含很少数据，也会将整个缓冲区写入映像。在小型数据库中，这表示相当一部分的映像可能为空。

还可以选择执行以下任何操作来缩短完成一次备份操作所需的时间：

- 增大 BACKUP DATABASE 命令中 PARALLELISM 参数的值，以便反映正在备份的表空间数。PARALLELISM 参数定义在压缩备份操作期间从数据库读取数据和压缩数据时，已启动的进程或线程数。将每个进程或线程分配给特定表空间，因此，为 PARALLELISM 参数指定的值大于要备份的表空间数并无益处。备份完此表空间后，会请求另一个表空间。但是应注意：每个进程或线程都需要内存和 CPU 开销。

- 增加备份缓冲区大小。理想的备份缓冲区大小是表空间扩展数据块大小的倍数加一页。如果有多个扩展数据块大小不同的表空间，那么将值指定为扩展数据块大小的公倍数加一页。
- 增加缓冲区的数量。使用的缓冲区至少是备份目标(或会话)的两倍，以确保备份目标设备时不必等待数据。
- 使用多个目标设备。

上面的这些参数，在数据库恢复的时候同样可以指定用来提高恢复性能。

1.4　DB2 内存体系结构

理解 DB2 的内存体系结构和 DB2 如何使用内存，可以防止过度分配内存，并有助于对内存的使用进行调优，从而获得更好的性能。

DB2 内存体系结构涉及如下内容：

- DB2 内存体系结构
- 内存集、内存池和内存块
- 内存自动调优

图 1-10 说明了 DB2 内存结构。这种内存结构在所有平台上都是一致的。

注意：

在多分区环境中，下面的图 1-10 适用于多分区实例中的每个分区。

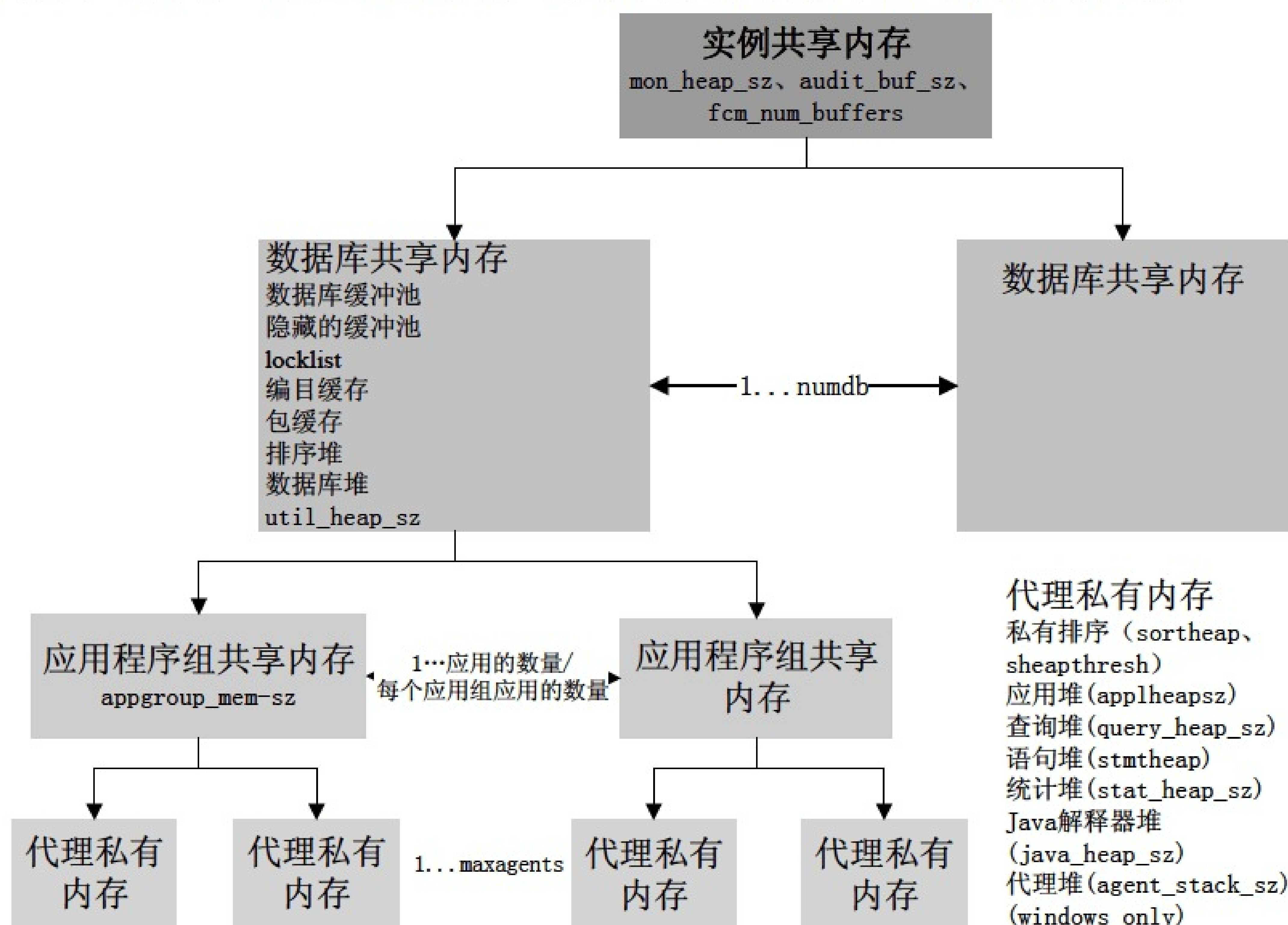


图 1-10　DB2 内存体系结构图

DB2 在 4 种不同的内存集(Memory Set)内拆分和管理内存。这 4 种内存集分别是：

- 实例共享内存(INSTANCE SHARE MEMORY)
- 数据库共享内存(DATABASE SHARED MEMORY)
- 应用程序组共享内存(APPLICATION GROUP SHARED MEMORY)
- 代理私有内存(AGENT PRIVATE MEMORY)

每种内存集由各种不同的内存池(也称堆)组成。图 1-10 还给出了各内存池的名称。例如，locklist 是属于数据库共享内存集的内存池。sortheap 是属于代理私有内存集的内存池。 下面我们将详细讨论每一种内存集。

1.4.1 实例共享内存

每个 DB2 实例都有一个实例共享内存。实例共享内存是在数据库管理器启动(db2start)时分配的，并随着数据库管理器的停止(db2stop)而释放。这种内存集用于实例级的任务，例如监控、审计和节点间通信，如图 1-11 所示。下面的数据库管理器配置(dbm cfg)参数控制着对实例共享内存以及其中个别内存池的限制：

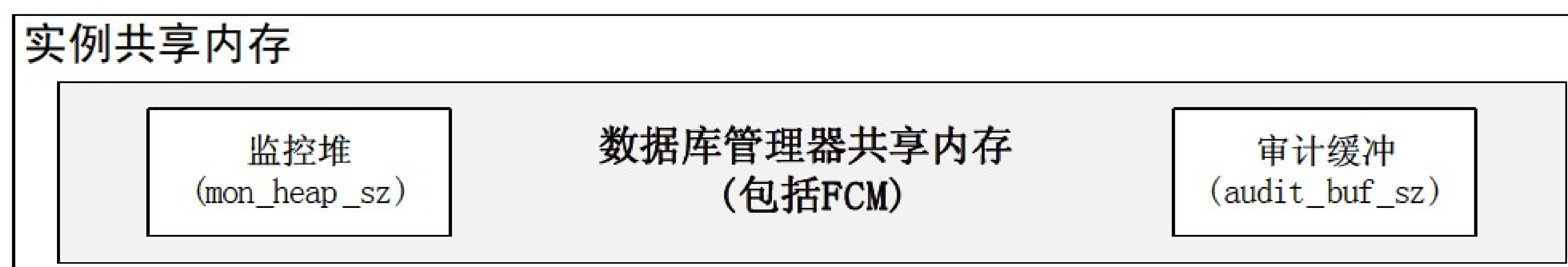


图 1-11 数据库管理器如何使用内存

- 实例内存(instance_memory)

此参数的默认值为 automatic。automatic 设置将导致在激活数据库分区时计算值并允许实例内存根据需要而增加，计算值介于系统上的物理 RAM 的 75%到 95%之间。系统越大，此百分比越高。对于具有内存使用量限制的 DB2 数据库产品，计算值也会受到产品许可证允许的最大值的限制。对于具有多个逻辑数据库分区的数据库分区服务器，此计算值是除以逻辑数据库分区数而获得的值。如果将内存使用量设置为特定值，那么此参数指定可以为数据库分区分配的最大内存量。

- 监视器堆(mon_heap_sz)

确定分配给数据库系统监视器的数据的内存量(以页计)。当进行数据库监视活动(例如生成快照、调整监视器开关、重新设置监视器或激活事件监视器)时，从监视器堆分配内存。

- 审计缓冲区(audit_buf_sz)：用于 DB2 AUDIT 实用程序。此内存区用于数据库审计活动。

● FAST COMMUNICATIONS BUFFERS(fcm_num_buffers)：用于分区之间的节点间通信，仅适用于多分区的实例。

性能相关建议

由于从 V9.5 开始，DB2 的数据库共享内存的大小受到了实例内存的限制，即数据库共享内存的大小不能超过实例内存。鉴于此，建议将实例内存根据需要设置为固定值以避免 DB2 内存使用量过大而造成系统性能下降。其他两个参数的值按照默认设置即可。

1.4.2　数据库共享内存

每个数据库都有一个数据库共享内存集。数据库共享内存是在数据库被激活(activate)或第一次被连接上的时候分配的。该内存集将在数据库处于非激活状态时释放(如果数据库先前处于激活状态)或者最后一个连接被断开的时候释放。这种内存用于数据库级的任务，例如备份/恢复、锁定和 SQL 的执行。

图 1-12 展示了数据库共享内存集内的各种内存池，括号中显示了控制这些内存池大小的配置参数。

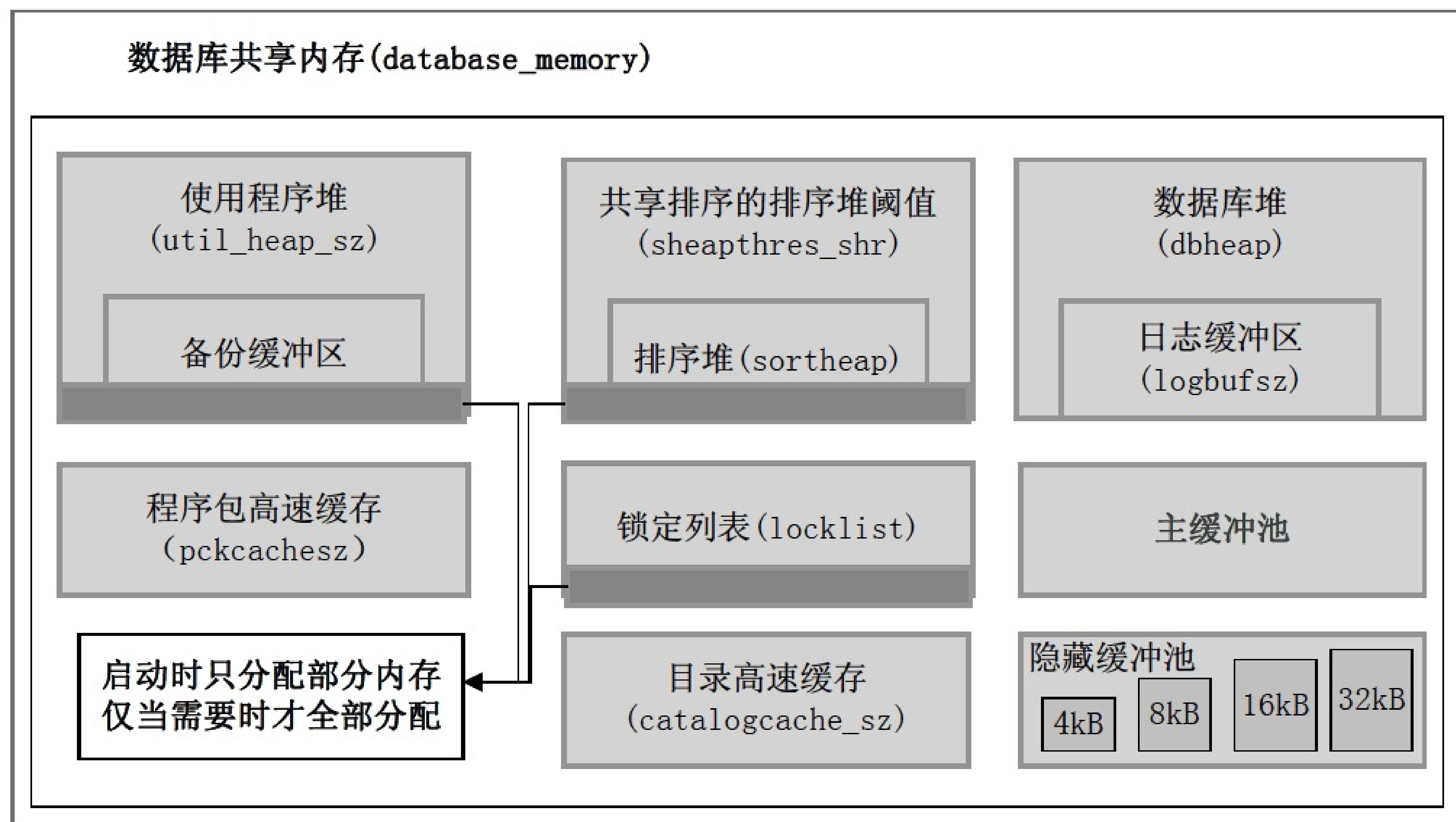


图 1-12　DB2 数据库共享内存

完整的方框意味着在数据库启动的时候，该内存池是完全分配的，否则就只分配部分的内存。例如，当数据库第一次启动时，不管 util_heap_sz 的值是多少，只有大约 16KB 的

内存被分配给实用程序堆。当数据库实用程序(例如备份、恢复、导出、导入和装载)启动时，才会按 util_heap_sz 指定的大小分配全额内存。下面对每个内存区域一一介绍：

1. 主缓冲池

数据库缓冲池通常是数据库共享内存中最大的一块内存。DB2 在其中缓冲所有常规数据和索引数据。一个数据库必须至少有一个缓冲池，并且可以有多个缓冲池，这要视工作负载的特征、数据库中使用的数据库页面大小等因素而定。例如，页面大小为 8KB 的表空间只能使用页面大小为 8KB 的缓冲池。

2. 隐藏的缓冲池

当数据库启动时，要分配 4 个页面大小分别为 4KB、8KB、16KB 和 32KB 的小型缓冲池。这些缓冲池是“隐藏”的，因为在系统编目中看不到它们(通过 SELECT * FROM SYSCAT.BUFFERPOOLS 显示不出来)。但是可以通过“db2 get snapshot”命令监控到，例如在下面的监控中我们就可以看到这 4 个隐藏的缓冲池：

```
$db2 get snapshot for bufferpools on sample |grep "Bufferpool name"
Bufferpool name                              = IBMDEFAULTBP
Bufferpool name                              = TESTBP
Bufferpool name                              = TESTBP2
Bufferpool name                              = BP32
Bufferpool name                              = IBMSYSTEMBP4K
Bufferpool name                              = IBMSYSTEMBP8K
Bufferpool name                              = IBMSYSTEMBP16K
Bufferpool name                              = IBMSYSTEMBP32K
```

隐藏的缓冲池的主要作用是：如果主缓冲池配置得太大，那么可能出现主缓冲池不适合可寻址内存空间的情况(不过在 64 位系统中很少碰到，除非操作系统的物理内存资源紧张)。这意味着 DB2 无法启动数据库，因为一个数据库至少必须有一个缓冲池。如果数据库没有启动，那么就不能连接到数据库，也就不能更改缓冲池的大小。由于这个原因，DB2 预先分配了 4 个这样的小型缓冲池。这样一来，一旦主缓冲池无法启动，DB2 还可以使用这些小型的缓冲池来启动数据库(在此情况下，用户将收到一条警告(SQLSTATE 01626))。这时，应该连接到数据库并减少主缓冲池的大小。

3. util_heap_sz

指示可由 BACKUP、RESTORE 和 LOAD(包括装入恢复)实用程序同时使用的最大内存量。

建议：使用默认值，除非实用程序耗尽空间，在这种情况下应增大此值。如果系统中的内存受约束，那么您可能期望降低此参数的值以限制数据库实用程序使用的内存。如果此参数设置得太低，您可能无法并行运行实用程序。应该根据需要动态更新此参数。如果实用程序较少，那么将此参数设置为较小的值。如果实用程序较多，或者实用程序消耗的内存较多，那么应该将此参数设置为较大的值。

4. pckcachesz

用于高速缓存数据库上的静态和动态 SQL 以及 XQuery 语句的部分。高速缓存程序包使数据库管理器在重新装入程序包时可以不访问系统目录；或者对于动态 SQL 或 XQuery 语句，可以免去编译这一步，从而减少其内部开销。将这些段保存在程序包高速缓存中，直到发生下列其中一种情况：

- 数据库关闭
- 程序包或动态 SQL 或 XQuery 语句无效
- 高速缓存空间用完

由 pckcachesz 参数指定的限制是软限制。假设数据库共享集中还有可用的内存，如果有必要，可以超过该限制。可使用 pkg_cache_size_top 监视元素确定程序包高速缓存达到的最大值，并用 pkg_cache_num_overflows 监视元素确定超过了由 pckcachesz 参数指定的限制的多少倍。

5. logbufsz

在将日志记录写入磁盘之前，此参数允许您指定用作这些记录的缓冲区的数据库堆大小(由 dbheap 参数定义)。

当发生下列一种情况时会将日志记录写入磁盘：

- 一个事务落实或一组事务落实，由 mincommit 配置参数定义
- 日志缓冲区已满
- 发生了某些其他内部数据库管理器事件

此参数也必须小于或等于 dbheap 参数。缓冲日志记录将使日志记录文件 I/O 更有效，因为将日志记录写入磁盘的频率越低，每次可写入的日志记录就越多。

建议：如果在专用的日志磁盘上有大量的读取活动，或者频繁使用磁盘，那么增大此缓冲区的大小。当增大此参数的值时，您也应考虑 dbheap 参数，因为该日志缓冲区使用由 dbheap 参数控制的空间。可以使用数据库系统监视器来确定将多少日志缓冲区空间用于特定事务(或工作单元)。参阅 log_space_used(使用的工作单元日志空间)监视元素。

6. catalogcache_sz

此参数指定目录高速缓存可以使用的数据库堆中的最大空间(以页计)。

高速缓存各个数据库分区中的目录信息允许数据库管理器不需要访问系统目录(或分区数据库环境中的目录节点)即可获取先前检测的信息，从而降低其内部开销。使用目录高速缓存可以帮助提高下列各项的整体性能：

- 绑定程序包以及编译 SQL 和 XQuery 语句
- 涉及检查数据库级别特权、例程特权、全局变量特权和角色权限的操作
- 连接至分区数据库环境中的非目录节点的应用程序

7. dbheap

此参数确定数据库堆使用的最大内存。

每个数据库都有一个数据库堆，并且数据库管理器代表所有连接至数据库的应用程序使用数据库堆。其中包含表、索引、表空间和缓冲池的控制块信息，还包括日志缓冲区的空间(logbufsz)和实用程序使用的临时内存。因此，堆大小将取决于许多变量。控制块信息保存在堆中，直到所有应用程序与数据库断开连接为止。启动数据库管理器时需要的最小量是在第一次连接时分配的。数据区将根据需要扩展，直到达到配置的上限，或者在设置为 automatic 的情况下，直到耗尽所有 database_memory 和/或 instance_memory 内存为止。

当决定要为 dbheap 配置参数指定值时，可以使用以下公式作为粗略准则：

10KB/表空间+ 4KB/表+(1KB + 4*使用的扩展数据块)/范围集群表(RCT)

配置的 dbheap 值仅表示分配的一部分数据库堆。数据库堆是用于满足数据库共享内存需求的主内存区。它将调整大小，以便除了包括 dbheap 值外，还包括启动数据库所需的基本分配值。用于报告内存使用情况的工具(如内存跟踪程序、快照监视器和 db2pd)将报告较大的那个数据库堆的统计信息。不会单独跟踪 dbheap 配置参数所表示的分配值。因此，这些工具所报告的数据库堆内存使用情况的统计信息超过为 dbheap 参数配置的值是很正常的。可以使用数据库系统监视器并借助 db_heap_top(分配的最大数据库堆)元素来跟踪用于数据库堆的最大内存量。

8. locklist

此参数指示分配给锁定列表的内存量。每个数据库都有一个锁定列表，锁定列表包含由同时连接至数据库的所有应用程序挂起的锁定。

在所有平台上，每个锁定需要 128 或 256 字节的锁定列表，这取决于是否对该对象挂起了其他锁定：

- 对于未挂起其他锁定的对象，挂起一个锁定需要 256 个字节。
- 对于存在挂起的锁定的对象，记录一个锁定需要 128 个字节。

一旦锁定列表已满，性能就可能会降低，因为锁定升级将生成更多的表锁定和更少的行锁定，从而降低数据库中共享对象的并行性。另外，应用程序之间可能有更多的死锁(因为这些应用程序都在等待有限数目的表锁定)，这样将导致事务回滚。当数据库的锁定请求数达到最大值时，应用程序将接收到值为-912 的 SQLCODE。

建议：如果锁定升级导致性能问题，那么可能需要增大此参数或 maxlocks 参数的值。可以使用数据库系统监视器来确定是否正在发生锁定升级。参阅 lock_escals(锁定升级)监视元素。

9. sortheap

如果排序为专用排序，那么此参数将影响代理程序专用内存。如果排序为共享排序，那么此参数将影响数据库共享内存。每个排序都有一个独立的排序堆，该排序堆由数据库管理器根据需要分配。此排序堆是将数据排序的区域。如果由优化器定向，那么将使用优化器提供的信息分配比此参数指定的排序堆小的排序堆。

建议：当使用排序堆时，应该考虑下列事项：

- 适当的索引可使排序堆的使用减至最小程度。
- 散列连接缓冲区、块索引 AND 运算、合并连接、内存中的表以及动态位映射(用于索引 AND 运算和星型连接)使用排序堆内存。在使用这些技术时，增大此参数的大小。
- 当需要进行频繁的大型排序时，增大此参数的大小。
- 当增大此参数的值时，应该检查是否还需要调整数据库管理器配置文件中的 sheapthres 和 sheapthres_shr 参数。
- 排序堆大小由优化器在确定访问路径时使用。在更改此参数之后，应考虑重新绑定应用程序(使用 REBIND 命令)。

10. sheapthres

对专用排序在任何给定时间可以使用的总内存量的实例范围软限制。当某个实例使用的专用排序内存总量达到此限制时，为其他传入专用排序请求分配的内存将显著减少。

建议：理想情况下，应将此参数设置为您在数据库管理器实例中拥有的最大 sortheap 参数的合理倍数。此参数至少应是该实例中为任何数据库定义的最大 sortheap 的两倍。

11. sheapthres_shr

表示对排序内存使用者每次可使用的数据库共享内存总量的软限制。

除排序以外，还有其他排序内存使用者(例如散列连接、索引 AND 运算、块索引 AND 运算、合并连接和内存表)。当共享排序内存使用者的共享排序总量达到 sheapthres_shr 限制时，就会激活内存调节机制，并且将来的共享排序内存使用者请求得到的内存量将少于请求的内存量，但始终多于完成任务所需的最低内存量。一旦达到 sheapthres_shr 限制，排序内存使用者的所有共享排序内存请求都将获得完成任务所必需的最低内存量。当活动共享排序内存使用者的共享内存总量达到此限制时，后续排序可能会失败(SQL0955C)。

当数据库管理器配置参数 sheapthres 的值为 0 时，数据库的所有排序内存使用者都将使用 sheapthres_shr 控制的数据库共享内存，而不是使用专用排序内存。

性能相关建议

评估数据库共享内存的分配大小并没有统一的方法，但是有如下经验供大家参考：

- 如果数据库服务器独占一台机器，可以把物理内存的 60%分配给数据库共享内存。
- 如果应用服务器和数据库服务器共享一台机器，可以把物理内存的 30%分配给数据库共享内存。

数据库共享内存由图 1-10 所示的内存使用者组成，在实际对数据库各个内存使用者评估大小的时候，同样没有精确的计算方法可对内存使用者一一做评估。为了省事，大家可以把数据库共享内存设置成内存自动调整，由于各内存使用者之间会动态地自动调整，因此不需要考虑各个参数值的设置。但是这样会造成潜在的性能问题，即各个内存使用者会根据需要随时进行内存调整，对于负载小的系统，内存调整量不是很大，影响较小；对于负载很大的系统，内存的调整值往往比较大，比如缓冲池调整，这样会对数据库的整体性能产生很大的影响。

所以，比较合理的方法就是先打开内存自动调整，运行一段时间后，把各内存使用者曾经使用到的最大值记录下来，然后关闭内存自动调整，把取出来的最大值赋给各个内存使用者作为初始固定值。这样既给出了各个内存使用者的合理参数值，又避免了内存调整带来的数据库性能影响。

1.4.3 应用程序共享内存

1. DB2 中的应用程序共享内存

DB2 中的应用程序共享内存的管理机制如图 1-13 所示。

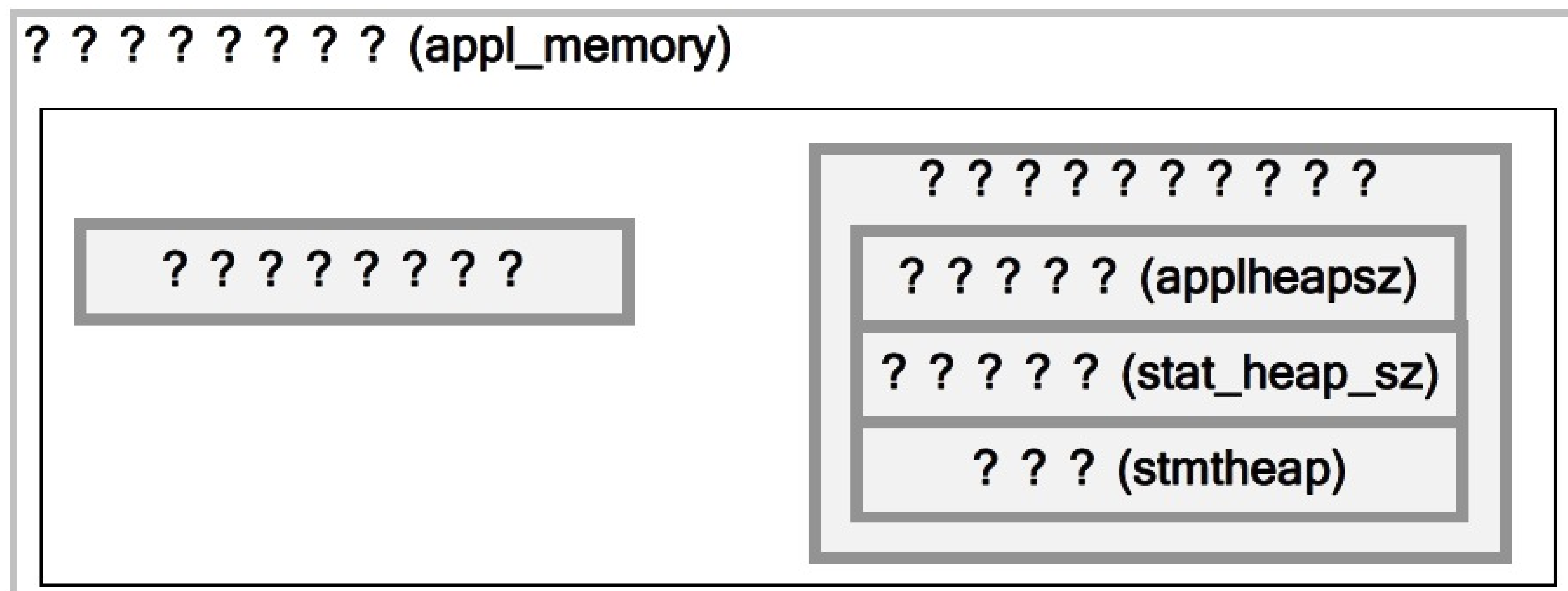


图 1-13　DB2 中应用程序共享内存的结构

下面对应用程序相关的内存作如下介绍：

- appl_memory

此参数允许 DBA 和 ISV 控制 DB2 数据库代理程序分配的用于为应用程序请求提供服务的最大应用程序内存量。在默认情况下，此参数的值设置为 automatic，这表示只要数据库分区分配的总内存量未超过 *instance_memory* 限制，就允许所有应用程序内存请求。

appl_memory 设置为 automatic 时，在数据库激活期间分配的初始应用程序内存最小，并且会根据需要增大(或减小)。将在内存中应用更改，并且 *appl_memory* 的值在磁盘上不会更改，与 db2 get db cfg show detail 命名显示的结果相同。在下一次激活时将重新计算此值。如果 *appl_memory* 设置为特定值，那么在数据库激活期间最初会分配请求的内存量，并且应用程序内存大小不会改变。如果不能从操作系统分配初始应用程序内存量，或初始应用程序内存量超过 *instance_memory* 限制，那么数据库激活将失败，并且出现 SQL1084C 错误(不能分配共享内存段)。

applheapsz

这是整个应用程序可以消耗的应用程序内存的总量，此参数定义最大应用程序堆大小。当应用程序第一次与数据库连接时，将为每个数据库应用程序分配应用程序堆。该堆将由为该应用程序工作的所有数据库代理程序共享(在先前发行版中，每个数据库代理程序都会分配自己的应用程序堆)。将根据需要从应用程序堆中分配用于处理应用程序的内存，最大为此参数指定的限制。将参数设置为 automatic 时，允许应用程序堆根据需要增长，直到到达数据库的 appl_memory 限制或数据库分区的 instance_memory 限制。当应用程序断开与数据库的连接时，将释放整个应用程序堆。

- stmtheap

在编译 SQL 或 XQuery 语句期间用作 SQL 或 XQuery 编译器的内存工作空间。此区域并不总是处于分配状态，但要对每个处理的 SQL 或 XQuery 语句进行分配和释放。注意：

对于动态 SQL 或 XQuery 语句，将在程序执行期间使用此工作区； 而对于静态 SQL 或 XQuery 语句，在绑定进程而不是在程序执行期间使用此工作区

- stat_heap_sz

指示使用 RUNSTATS 命令收集统计信息时所用的最大堆大小。

RUNSTATS 内存需求取决于几个因素。统计信息选项越多，使用的内存也就越多。例如，如果正在收集 LIKE 统计信息或 DETAILED 索引统计信息，那么收集列统计信息时，收集的列统计信息的数目越多，使用的内存也越多；收集分布统计信息时，收集的频率和/或分位数值越多，使用的内存也越多。建议使用默认设置 automatic。

2. 性能相关建议

在实际的运维中，多数环境不会限制连接到数据库的应用程序个数，而每个连接到数据库的应用程序都会不同程度地用到内存、如果应用程序共享内存不足，程序运行就会失败。所以建议把应用程序相关的如上参数的值都设置成 automatic 以支持系统正常运行.

1.4.4 代理私有内存

每个 DB2 代理进程都需要获得内存以执行其任务。代理进程将代表应用程序使用内存来优化、构建和执行访问计划，执行排序，记录游标信息(例如位置和状态)，收集统计信息等。为响应并行环境中的连接请求或新的 SQL 请求，要为 DB2 代理分配代理私有内存。

代理的数量受下面两者中的较低者限制：

- 所有活动数据库的数据库配置参数 maxappls 的总和，这指定了允许的活动应用程序的最大数量。
- 数据库管理器配置参数 maxagents 的值，这指定了允许的最大代理数。

代理私有内存集由以下内存池组成。这些内存池的大小由括号中的数据库配置参数指定：

- application heap(applheapsz)
- sort heap(sortheap)
- statement heap(stmtheap)
- statistics heap(stat_heap_sz)
- query_heap(query_heap_sz)
- java heap size(java_heap_sz)
- agent stack size(agent_stack_sz) (仅适用于 windows)

代理私有内存是在 DB2 代理被“指派”执行任务时分配给 DB2 代理的。那么，私有内存何时释放呢？答案取决于 DBM CFG 参数 num_poolagents 的值。该参数的值指定任何

时候可以保留的空闲代理的最大数目。如果该值为 0，那么就不允许有空闲代理。只要有代理完成工作，这个代理就要被销毁，它的内存也要返回给操作系统。如果该参数被设为非零值，那么代理在完成其工作后不会被销毁。相反，它将被返回到空闲代理池，直到空闲代理的数目到达 num_poolagents 指定的最大值。当传入新的请求时，就要调用这些空闲代理来服务这个新的请求。这样就减少了创建和销毁代理的开销。

当代理变成空闲代理时，它仍然保留了其代理的私有内存。这样设计是为了提高性能，因为当代理被再次调用时，它便有准备好的私有内存。如果有很多的空闲代理，并且所有这些空闲代理都保留了它们的私有内存，那么就可能导致系统耗尽内存。为了避免这种情况，DB2 使用一个注册变量来限制每个空闲代理可以保留的内存量。这个变量就是 DB2MEMMAXFREE，它的默认值是 8 388 608 字节。这意味着每个空闲代理可以保留最多 8MB 的私有内存。如果有 100 个空闲代理，那么这些代理将保留 800MB 的内存，因此它们很快就会耗尽 RAM。您可能希望降低或增加这一限制，这取决于 RAM 的大小。图 1-14 展示了在同一系统上有两个实例并发运行的情况。虚拟内存包括物理 RAM 和调页空间 (page space)。共享内存“倾向于”留在 AM 中，因为对它们的访问更频繁。如果代理空闲了较长的一段时间，那么代理私有内存将被调出。

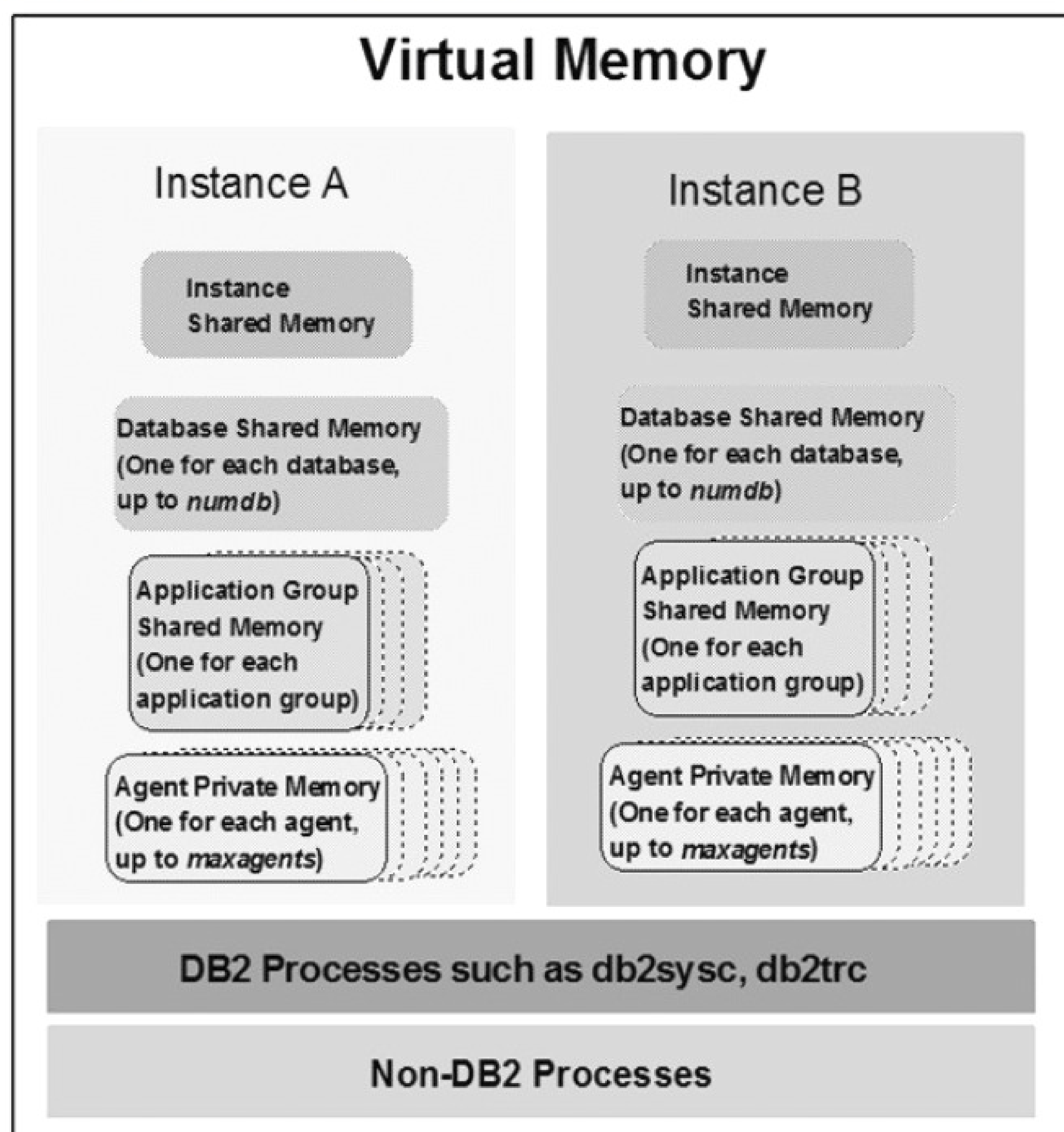


图 1-14 并发运行的两个 DB2 实例的内存分配图

性能相关建议

针对代理私有内存，除非发现明显的性能问题和需要调整的必要，否则不建议修改默认值。

1.4.5 代理程序与应用程序之间通信时的内存

代理程序和应用程序之间的通信分为以下两种不同情况：

- 本地应用程序与代理通信
- 远程应用程序与代理通信

本地应用程序与代理通信时使用本机上的共享内容，其大小由参数 aslheapsz 确定。如果对数据库管理器的请求或与之相关联的应答不适合该缓冲区，那么请求和应答将分成两个或更多的发送-接收对。应将此缓冲区的大小设置为可使用单个发送-接收对来处理大多数请求。

当客户机请求与远程数据库连接时，在客户机上分配通信缓冲区。在数据库服务器上，最初分配大小为 32767 字节的通信缓冲区，直到建立连接从而服务器可确定客户机上 RQRIOBLK 的值为止。一旦服务器知道此值，如果客户机缓冲区不为 32767 字节，服务器就将重新分配通信缓冲区。

1.4.6 共享内存与私有内存

至此，我们已经讨论了实例共享内存、数据库共享内存、应用程序组共享内存、应用程序共享内存以及代理私有内存。但是共享内存和私有内存的区别和意义是什么呢？

我们知道，所有数据库请求都是由 DB2 代理或子代理来服务的。例如，当有应用程序连接到数据库时，就有 DB2 代理指派给它。当应用程序发出任何数据库请求(例如 SQL 查询)时，DB2 代理就会出来执行完成这个查询所需的所有任务——代表应用程序工作(如果数据库是分区的，或者启用了 intra_parallel，那么可以分配不止一个代理来代表应用程序工作。这些代理叫作子代理)。每个代理或子代理都被当作 DB2 进程，通过获得一定数量的内存来执行工作。这种内存被称作代理私有内存——不能与其他任何代理共享。在前面我们曾提到过，代理私有内存包括一些内存池，例如应用程序堆大小、排序堆大小和语句堆大小。

除了私有内存(代理在其中使用排序堆执行“私有”任务，例如私有排序)外，代理还需要数据库级的资源，例如缓冲池、locklist 和日志缓冲区。这些资源在数据库共享内存中。DB2 的工作方式是，数据库共享内存中的所有资源都由连接到相同数据库的所有代理或子

代理共享。因此，该内存集被称作共享内存而不是私有内存。例如，连接到数据库 A 的代理 X 使用数据库 A 的数据库共享内存中的资源。现在又有代理 Y 也连接到数据库 A。那么代理 Y 将与代理 X 共享数据库 A 的数据库内存(当然，代理 X 和代理 Y 都有它们自己的代理私有内存，这些代理私有内存不是共享的)。

这样的逻辑同样适用于实例共享内存和应用程序组共享内存。

图 1-15 展示了当两个 DB2 代理(代理 X 和代理 Y)连接到数据库 A 时分配的 DB2 内存集。假设：

- 数据库 A 属于实例 DB2INST1。
- 数据库 A 为应用程序组 1 启用了 intra_parallel。
- 代理 X 和代理 Y 都属于应用程序组 1。

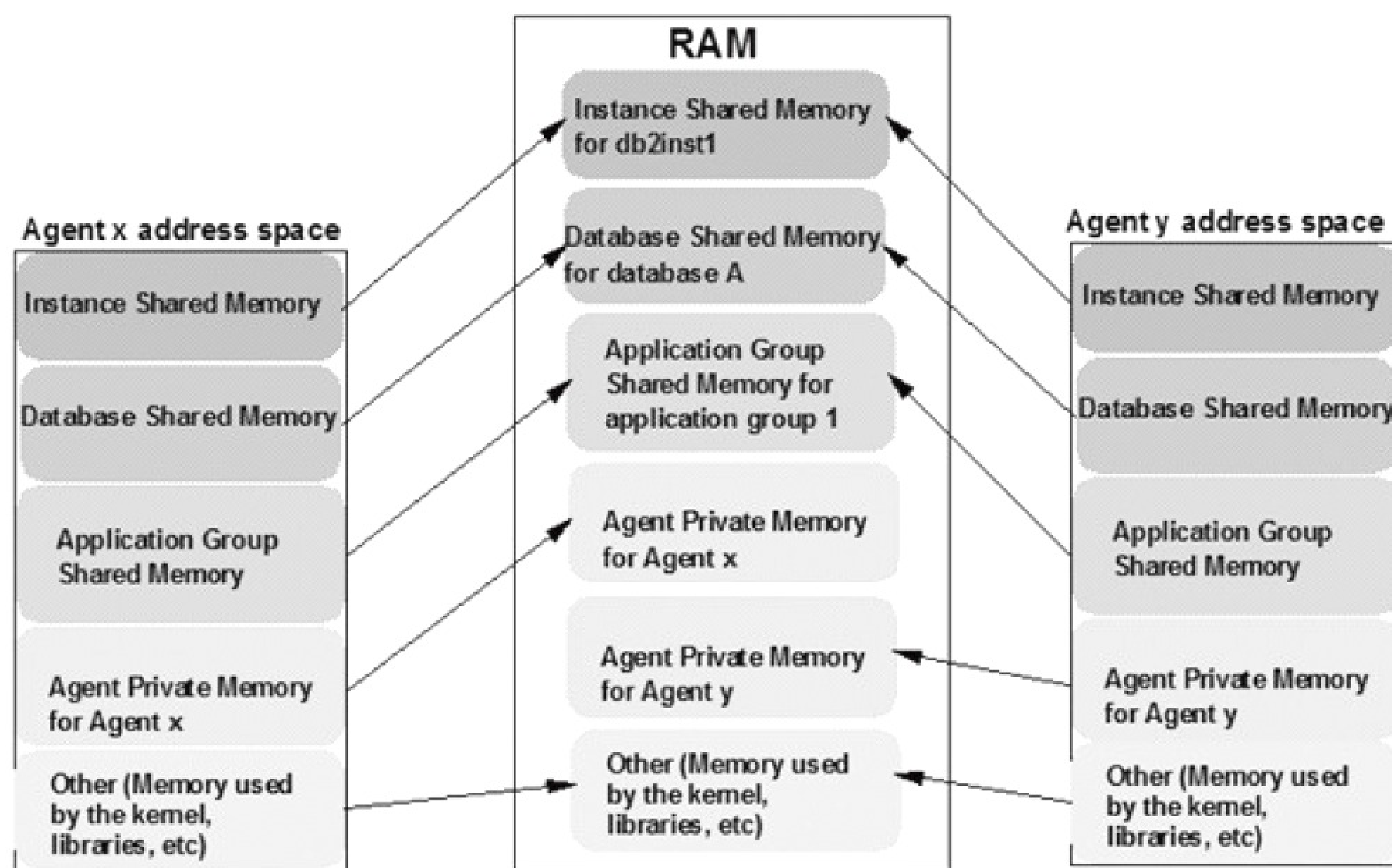


图 1-15　DB2 代理进程内存地址空间

图 1-15 展示了在 RAM 中分配的以下内存集：

- 用于实例 DB2INST1 的实例共享内存集。
- 用于数据库 A 的数据库共享内存集。
- 用于应用程序组 1 的应用程序组共享内存。

- 用于代理 X 的代理私有内存集。
- 用于代理 Y 的代理私有内存集。
- 为内核和库等预留的内存。

代理 X 和代理 Y 共享相同的实例内存、数据库内存和应用程序组内存，因为它们属于相同的实例、相同的数据库和相同的应用程序组。此外，它们有自己的代理私有内存。每个 DB2 代理进程都有自己的内存地址空间。在内存空间中的内存地址允许代理访问物理 RAM 中的内存。我们可以把这些地址看作指向 RAM 的指针，如图 1-15 所示。对于任何 DB2 进程，这个地址空间必须能够容纳上述所有 4 种内存集。

1.5 内存集、内存池和内存块

DB2 中的操作系统服务(OSS)组件向其他组件提高了三个级别的抽象内存管理，包括内存集(Memory Set)、内存池(Memory Pool)、内存块(Memory Block)。Memory Block 是一片连续的内存空间，是 DB2 内部内存分配的基本单位。内存块是一段连续的内存地址范围，主要用于被其他组件申请。DB2 由内部结构跟踪内存块分配情况，一个内存块只能属于一个内存池。内存池由若干个内存块组成，其地址空间并不连续。内存池跟踪与其相关的内存块总和，并且控制其值不能超过池创建时指定的大小。每个内存池都属于一个内存集，这意味着属于同一内存池的内存块都从该内存集中分配。Memory Pool 是“DB2 Memory Manager”分配给不同组件的一组内存。DB2 根据内存使用的范围、用途、行为来组织分配，所以有了 Memory Pool 的概念。有些 Memory Pool 只为单个用途服务，比如 package cache；而有些 Memory Pool 会为多个用途服务，比如 dbheap、application heap 等。Memory Set 是从操作系统的角度来对内存分配进行管理。内存集是机器内存的实际空间，占用操作系统分配给各个进程(DB2 的各个进程在操作系统看来都是普通进程)的地址空间。内存集可以是共享的，也可以是进程私有的。共享的内存集在操作系统上表现为共享内存段。

Share Memory Set(共享内存集)在 UNIX 上会对应到一个或多个 IPC 共享内存段。Private Memory Set(私有内存集)由多个 Private Memory Set Applications 组成。一个 Memory Set 可以包含多个 Private Memory Pool，一个 Memory Pool 只能属于一个 Memory Set。Memory Set、Memory Pool 和 Memory Block 的关系如图 1-16 所示。

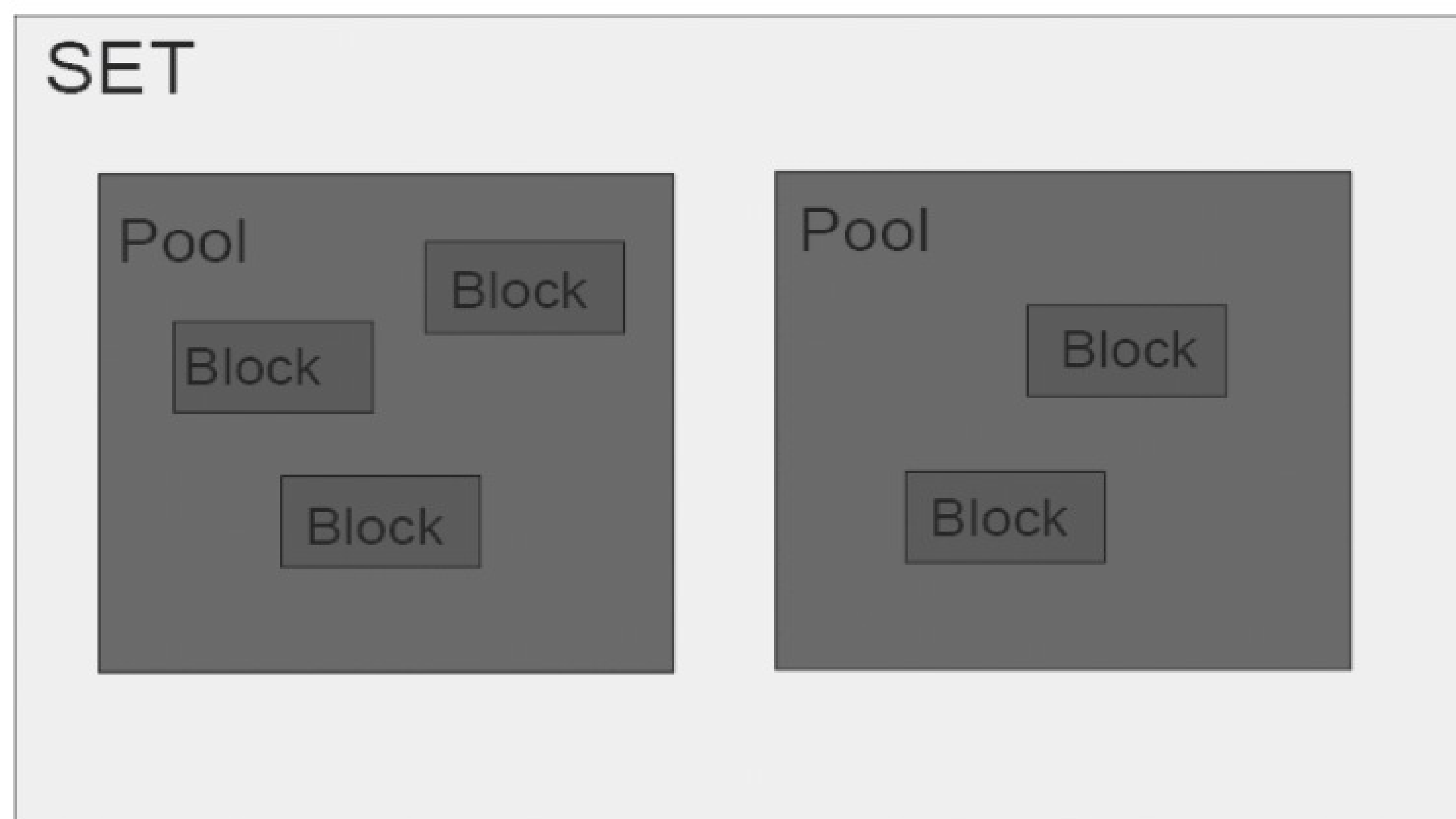


图 1-16　Memory Set、Memory Pool 和 Memory Block 示意图

在本章的前面我们讲过，在 DB2 中主要存在实例共享内存、数据库共享内存、应用程序组共享内存和代理私有内存这 4 类内存集。

1.5.1　实例级内存集

我们可以使用 db2pd 工具浏览 DB2 内存结构，下面是实例级内存集的输出结果，如下所示：

```
    $ db2pd -inst -memsets
    Database Partition 0 -- Active -- Up 0 days 00:01:09
    Memory Sets:
    Name          Address     Id          Size(Kb)   Key          DBP    Type
Unrsv(Kb)  Used(Kb)   Cmt(Kb)    Uncmt(Kb)
    DBMS          0x10000000 1769475     32320      0x90918C61   0      0      5056
11264      11264      21056
    FMP           0x11F90000 1802246     22592      0x0          0      2      0
192        22592      0
    Trace         0x00000000 1736706     8487       0x90918C74   0      -1     0
8487       8487       0
```

关于输出的每个字段的详细解释大家可以查看 DB2 信息中心，我们重点关注以下列：

- Name：内存集的名称。
- Id：内存集的标识。
- Size(Kb)：内存集的总大小，为 Cmt(Kb)与 UnCmt(Kb)之和。
- Unrsv(Kb)：是指未预留给特定内存池的大小，任何内存池都可以使用这部分。

- Used(Kb)：是当前已经分配给内存池的大小。
- Cmt(Kb)：是 DB2 已经向操作系统提交的内存。这些内存既可以是物理内存，也可以是交换空间。
- Uncmt(KB)：当前还没有使用、但被 DB2 标识为未提交的内存。根据操作系统的不同，这部分内存既可以是物理内存，也可以是交换空间。

上面的输出显示 DB2 在实例级别有三个内存集，依次为实例使用(DBMS)、保护存储过程使用(FMP)、跟踪工具使用(Trace)的内存集。这些内存集都是共享的，可以通过操作系统的“ipcs”命令看到，结果输出如下所示：

```
$ ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 65537      db2inst1   767        8691272    1          dest
0x90918c74 1736706    db2inst1   767        8691272    7
0x90918c61 1769475    db2inst1   701        33095680   5
0x00000000 1933316    db2inst1   701        131072     2
0x00000000 1867781    db2inst1   701        41025536   1
0x00000000 1802246    db2fenc1   701        23134208   3
0x00000000 1900551    db2inst1   701        119209984  1
0x00000000 1409032    db2inst1   701        131072     1          dest
0x00000000 2949129    db2inst1   701        131072     2
```

大家可以注意到，=“ipcs”命令输出的 shmid、key 与 db2pd 命令输出的 id、key 相同，“ipcs”输出的 bytes 输出的 db2pd 是 SIZE(Kb)转换为字节单位后的结果。

在下面的 db2pd 输出中，最大的 DBMS 部分为实例共享内存，其大小有实例参数 instance_memory 控制。

如果希望进一步了解 DBMS 的详细使用情况，可以使用下面的命令：

```
    $ db2pd -inst -mempools
    Database Partition 0 -- Active -- Up 0 days 00:01:15

    Memory Pools:
    Address    MemSet    PoolName    Id    Overhead    LogSz       LogUpBnd
LogHWM      PhySz       PhyUpBnd    PhyHWM      Bnd BlkCnt CfgParm
    0x10000A2C DBMS     fcm         74    0           0           608414      0
0           655360      0           Ovf 0       n/a
    0x10000980 DBMS     fcmsess     77    65440       845168      1118208
845168      983040      1179648     983040      Ovf 3       n/a
    0x100008D4 DBMS     fcmchan     79    65440       159488      405504
```

```
159488      327680      458752      327680      Ovf 3      n/a
    0x10000828 DBMS     fcmbp      13    65440      590592      860160
590592      786432      917504      786432      Ovf 3      n/a
    0x1000077C DBMS     fcmctl     73    186304     1176257     3073620
1176257     1376256     3080192     1376256     Ovf 11     n/a
    0x100006D0 DBMS     monh       11    122592     143887      368640
143887      327680      393216      327680      Ovf 19     MON HEAP SZ
    0x10000624 DBMS     resynch    62    26928      104080      1703936
104080      196608      1703936     196608      Ovf 2      n/a
    0x10000578 DBMS    apmh       70    2672       441856      4128768     441856
458752      4128768     458752      Ovf 22     n/a
    0x100004CC DBMS     kerh       52    48         206936      3866624     206936
262144      3866624     262144      Ovf 50     n/a
    0x10000420 DBMS     bsuh       71    65408      2213972     9830400
2216928     2293760     9830400     2293760     Ovf 35     n/a
    0x10000374 DBMS     sqlch      50    0          1492576     1572864
1492576     1572864     1572864     1572864     Ovf 203    n/a
    0x100002C8 DBMS     krcbh      69    0          46072       65536       46072
65536       65536       65536       Ovf 12     n/a
    0x1000021C DBMS     eduah      72    1904       2816016     2816048
2816016     2818048     2818048     2818048     Ovf 1      n/a
    0x11F9021C FMP      undefh     59    8048       122900      22971520
122900      131072      23003136    131072      Phy 1      n/a
```

大家可以注意到，DBMS 内存集中包含 13 个内存池，FMP 内存集中包含 1 个内存池。在关于内存池的输出中我们重点关注以下列：

- PoolName：内存池的名称。
- Id：内存的标识。
- LogSz：内存池的逻辑大小。
- LogUpBnd：逻辑大小上限。
- LogHwm：逻辑高水位。
- PhySz：内存池的物理大小。
- PhyUpBnd：物理大小上限。
- PhyHwm：物理高水位。
- BlkCnt：内存块的个数。

其中，LogSz 指示的是与当前内存池对应的总内存块逻辑大小，PhySz 指的是与当前内存池对应的总内存块物理大小。已经获得的物理大小不一定都在机器的 RAM 中，因为一般操作系统都使用了 VMM，使得内存池可能还使用磁盘交换空间。这样的话，物理大小就可能大于逻辑大小。

我们可以对某个内存池继续使用 db2pd 工具来跟踪与其相关的内存块情况，下面的命令输出显示的是关于 ID 为 11 的监控堆块的使用情况：

```
    $ db2pd -memblocks 11

    Memory blocks in DBMS set for pool 11.

    Address    PoolID     PoolName    BlkAge Size(Bytes)  I LOC   File
    0x10AEEFB8 11         monh        8     69648        1 842   3400785722
    -----------------------------------------------省略
--------------------------------
    0x10AB1448 11         monh        20    26           1 533   1584961743
    Total size for DBMS memory set: 143819 bytes

    Memory blocks sorted by size for monh pool:
    PoolID     PoolName   TotalSize(Bytes)     TotalCount  LOC   File
    11        monh      139296              2          842   3400785722
    -----------------------------------------------省略
--------------------------------
    11        monh      8                  1         494   3284019182
    Total size for monh pool: 143819 bytes

    Total size for DBMS memory set: 143819 bytes

    All memory consumers in DBMS memory set:
    PoolID     PoolName   TotalSize(Bytes)     %Bytes TotalCount %Count LOC
File
    11        monh      139296              96.86  2          1.56   842
3400785722
    -----------------------------------------------省略
--------------------------------
    11        monh      8                  0.01   1          0.78   494
3284019182
```

memblock 命令输出的第一部分显示的是在 DBMS 内存集中的块分配情况(限定只显示 ID 为 11 的内存池的相关情况)，第二部分是为每个内存池按照块大小排序的块情况，第三部分显示的是各种块所占的百分比。

1.5.2 跟踪内存使用

通过定期收集内存集、内存池、内存块的个数和使用大小，我们可以获得数据库系统中在各个不同时间段的内存事情。这对于我们分析 DB2 内存增长的趋势、不同工作负载对内存的要求十分有用。

下面我们以监控堆内存使用为例，示范如何跟踪内存集、内存池、内存块的使用。

我们知道在 DB2 中，实例基本上有以下几个参数用于控制应用程序连接到实例后可以获取的监控数据有多少。这些监控数据使用监控堆作为临时存放地点，而监控堆是在实例内存集中分配的，大小由参数 mon_heap_sz 控制。查看 DBM CFG，如下所示：

```
$ db2 get dbm cfg show detail
------------------------------省---------------------------------------
 Transaction processor monitor name       (TP_MON_NAME) =
   Buffer pool                          (DFT_MON_BUFPOOL) = ON
   Lock                                  (DFT_MON_LOCK) = ON
   Sort                                  (DFT_MON_SORT) = ON
   Statement                              (DFT_MON_STMT) = ON
   Table                                (DFT_MON_TABLE) = ON
   Timestamp                          (DFT_MON_TIMESTAMP) = ON
   Unit of work                             (DFT_MON_UOW) = ON
 Database monitor heap size (4KB)          (MON_HEAP_SZ) = AUTOMATIC(90)
------------------------------省略-------------------------------------
```

当没有应用连接到实例时，监控堆分配的情况，如下所示：

```
$ db2 list applications
SQL1611W  No data was returned by Database System Monitor.

$ db2pd -inst -mempool
Database Partition 0 -- Active -- Up 0 days 06:30:02

Memory Pools:
Address    MemSet   PoolName   Id    Overhead   LogSz       LogUpBnd
LogHWM      PhySz       PhyUpBnd    PhyHWM      Bnd BlkCnt CfgParm
----------------------------------------------------------------------
x100006D0 DBMS      monh       11    122592     143819      368640      145359
327680      393216      327680      Ovf 16      MON_HEAP_SZ
```

目前，监控堆的物理大小为 327680 字节，逻辑大小为字节 143819，内存块的个数为 16。当应用程序连接到数据库时监控堆的大小没有发生变化，结果如下所示：

```
$ db2 connect to tp1
   Database Connection Information
 Database server        = DB2/LINUX 9.7.6
 SQL authorization ID   = DB2INST1
 Local database alias   = TP1
$ db2pd -inst -mempool
Database Partition 0 -- Active -- Up 0 days 06:30:02
```

```
    Memory Pools:
    Address     MemSet    PoolName   Id    Overhead   LogSz       LogUpBnd
LogHWM      PhySz       PhyUpBnd    PhyHWM      Bnd BlkCnt CfgParm
    ----------------------------------------------------------------------
    x100006D0 DBMS     monh       11    122592     143819      368640      145359
327680      393216      327680      Ovf 16      MON_HEAP_SZ
```

当应用程序执行 GET SNAPSHOT 语句后，监控堆的大小发生了变化，结果如下所示：

```
    $ db2 get snapshot for database on tp1 >/dev/null
    $ db2pd -inst -mempool
    Database Partition 0 -- Active -- Up 0 days 06:30:02

    Memory Pools:
    Address     MemSet    PoolName   Id    Overhead   LogSz       LogUpBnd
LogHWM      PhySz       PhyUpBnd    PhyHWM      Bnd BlkCnt CfgParm
    ----------------------------------------------------------------------
    0x100006D0  DBMS     monh       11    122592     143999      368640
145359      327680      393216      327680      Ovf 18      MON_HEAP_SZ
```

目前，监控堆的物理大小为 327680 字节，逻辑大小为 143999 字节，内存块的个数为 18。与没有应用程序连接时对比，逻辑大小增加了 180 字节，块的个数增加了 2，物理大小没有变化。而当应用程序断开后，监控堆的逻辑大小恢复到为 143819 字节，块的个数恢复到 16。

上面的实验表明每个新应用程序连接上来后，在没有获取监控数据时，不会使用监控堆，只有当执行了 GET SNAPSHOT 语句后才监控内存使用。当应用程序断开后，使用的监控堆又释放了。

通过上面监控堆的例子，我们看出通过对内存集、内存池、内存块的跟踪分析，可以发现 DB2 组件什么时候向操作系统服务(OSS)组件申请内存，什么时候释放内存。

1.5.3 定位内存泄漏

一般来说内存集、内存池、内存块之间存在以下关系：

- 内存集已用大小一般等于与其关联的内存池所有物理大小之和。
- 内存池的物理大小一般会大于其逻辑大小。
- 内存池的逻辑大小等于所有内存块的大小之和。
- 内存集的“Size – Unrsv”等于内存集相关所有内存池的“PhyUpBnd”之和。

根据上面这些关系，特别是第一个关系，可以帮助我们定位一些 DB2 OSS 组件报告的

内存方面的问题。

当 db2diag.log 中出现以下错误时，表明 OSS 从内存集中为某些内存池分配内存失败：

```
FUNCTION: DB2 UDB, oper system services, getPrivateChunksFromOs, probe:100
CALLED : OS, -, VirtualAlloc
OSERR : 8 "Not enough storage is available to process this command."
MESSAGE : Private memory and/or virtual address space exhausted
```

这时我们可以查看具体的内存集的已经使用大小、操作系统已经分配给 DB2 的大小，同时对相关内存池的物理大小求和，看内存池的物理大小之和是否明显小于内存集已使用大小。如果存在上述情况，就表明存在内存泄漏。换言之，在从内存集中分配内存至内存池后，当内存池删除时并没有将这些内存返回给内存集。

1.5.4　数据库级内存集

下面显示了数据库级内存集，输出结果如下：

```
    $ db2pd -memsets -db tp1
    Database Partition 0 -- Database TP1 -- Active -- Up 0 days 00:00:21

    Memory Sets:
    Name         Address    Id          Size(Kb)   Key        DBP    Type
Unrsv(Kb)  Used(Kb)   Cmt(Kb)    Uncmt(Kb)
    TP1         0xAB9CA000 1900551     116416     0x0        0      1      19072
17152      17152      99264
    AppCtl       0xB2C7A000 1867781     40000      0x0        0      12     0
512        512        39552
    App12       0x001D8004 1933316     128        0x0        0      4      0
128        128        0
    App11       0x00000000 0          0          0x0        0      0      0
0          0          0
    App10       0x00000000 0          0          0x0        0      0      0
0          0          0
    App9        0x00000000 0          0          0x0        0      0      0
0          0          0
    App8        0x00000000 0          0          0x0        0      0      0
0          0          0
```

其中 ID 为 0 的不会在“ipcs”中显示出来，表明相关应用或代理的共享内存已经不存在。名称为 TP1 的内存集为数据库共享内存，由数据库参数 DATABASE_MEMORY 决定，可以通过以下命令查看，如下所示：

```
    [db2inst1@unixhost ~]$ db2 get db cfg  for tp1 show detail |grep DATABASE_MEMORY
     Size of database shared memory (4KB)  (DATABASE_MEMORY) = COMPUTED(29088)
```

同样，可以通过执行操作系统命令“ipcs”查看数据库级的共享内存，如下所示：

```
$ ipcs -m
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 65537      db2inst1   767        8691272    1          dest
0x90918c74 1736706    db2inst1   767        8691272    7
0x90918c61 1769475    db2inst1   701        33095680   5
0x00000000 1933316    db2inst1   701        131072     2
0x00000000 1867781    db2inst1   701        41025536   1
0x00000000 1802246    db2fenc1   701        23134208   3
0x00000000 1900551    db2inst1   701        119209984  1
0x00000000 1409032    db2inst1   701        131072     1          dest
0x00000000 2949129    db2inst1   701        131072     2
```

可以使用 db2pd 命令显示所有数据级的内存池，如下所示：

```
$ db2pd -mempool -db tp1
Database Partition 0 -- Database TP1 -- Active -- Up 0 days 00:00:30

Memory Pools:
Address    MemSet   PoolName   Id    Overhead   LogSz       LogUpBnd
LogHWM      PhySz       PhyUpBnd    PhyHWM      Bnd BlkCnt CfgParm
0xAB9CAD88 TP1      utilh      5     0          2120        20512768    2120
65536       20512768    65536       Ovf 10      UTIL_HEAP_SZ
0xAB9CAC30 TP1      pckcacheh  7     57744      85520       Unlimited   85520
196608      Unlimited   196608      Ovf 2       PCKCACHESZ
0xAB9CAB84 TP1      xmlcacheh  93    50944      80008       20971520    80008
131072      20971520    131072      Ovf 1       n/a
0xAB9CAAD8 TP1      catcacheh  8     0          59488       Unlimited   59488
65536       Unlimited   65536       Ovf 9       CATALOGCACHE_SZ
0xAB9CA980 TP1      bph        16    32         1310384     Unlimited
1310384     1376256     Unlimited   1376256     Ovf 9       n/a
0xAB9CA828 TP1      bph        16    32         782592      Unlimited   782592
851968      Unlimited   851968      Ovf 5       n/a
0xAB9CA6D0 TP1      bph        16    32         520448      Unlimited   520448
589824      Unlimited   589824      Ovf 3       n/a
0xAB9CA578 TP1      bph        16    32         389376      Unlimited   389376
458752      Unlimited   458752      Ovf 2       n/a
0xAB9CA420 TP1      bph        16    32         323840      Unlimited   323840
393216      Unlimited   393216      Ovf 2       n/a
0xAB9CA374 TP1      shsorth    18    0          0           40960000    0
0           40960000    0           Ovf 0       SHEAPTHRES_SHR
```

```
    0xAB9CA2C8 TP1      lockh       4     32         328192      458752      328192
393216      458752      393216      Ovf 1      LOCKLIST
    0xAB9CA21C TP1      dbh        2     378080     12287604    24510464
12287604    12976128    24510464    12976128    Ovf 586    DBHEAP
    0xB2C7A2C8 AppCtl   apph        1      0          7602        1048576     17960
65536       1048576     65536       Phy 19     APPLHEAPSZ
    -----------------------------略------------------------------------------
    0xB2C7A21C AppCtl   appshrh     20    2208        46544       20480000    46544
131072      20512768    131072      Phy 26     application shared
```

其中，大家可以看到有名为 BPH 的内存池，这就是 DB2 缓冲池使用的内存。4 个为 DB2 隐藏缓冲池，页面大小分别为 4KB、8KB、16KB 和 32KB，另一个为创建数据库时默认创建的缓冲池 IBMDEFAULTBP。“CfgParm”列显示了与内存池相关的配置参数，如名为 DPH 的内存相关配置参数为 DBHEAP，表明这是数据库堆。我们在实例级别观察到的内存集、内存池、内存块之间的关系同样适用于数据库级共享内存，也适用于应用程序组共享内存。

1.6　内存自动调优

从 DB2 V9 开始，一种新的内存自动调优特性(自动调优内存管理)通过自动设置几个内存配置参数值的方式，简化了内存配置任务。内存自动调优一旦启用，就能够在内存使用者(包括排序、程序包缓存、锁定列表和缓冲池等)之间动态分配可用内存资源。内存自动调优在 DATABASE_MEMORY 配置参数定义的内存限制范围之内。DATABASE_MEMORY 的值本身可在 Windows 和 AIX 上自动调优。当启用针对 DATABASE_MEMORY 的内存自动调优之后(通过将其设置成 automatic)，DB2 内存自动调优组件会确定数据库的总内存需求，并会根据当前的数据库需求增加或减少分配给数据库共享内存的容量。例如，如果当前的数据库需求很高，而系统又有足够的空闲内存，数据库内存就可以使用较多的内存资源。一旦数据库内存的需求降低，或系统的空闲内存量过低，一部分数据库共享内存就会被释放。

图 1-17 描述了 DB2 内存自动调优影响的范围。

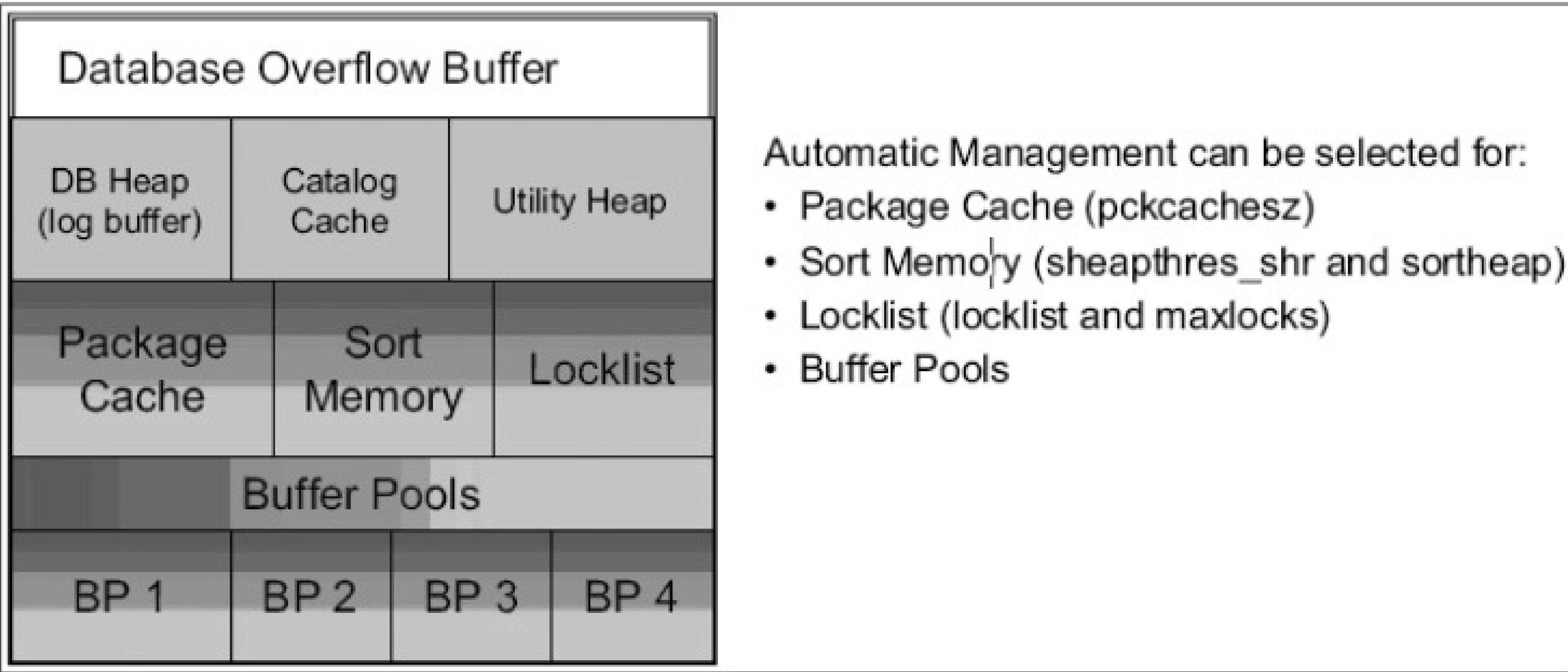


图 1-17　DB2 内存自动调优的影响范围

对于 DBA 而言，多个缓冲池的优化调整是比较困难的，内存自动调优进程基于当前工作负载动态增加或减少缓冲池大小。当缺少锁内存时，会引起数据库的锁提升，导致数据库系统性能下降。STMM 进程能够动态设置 locklist 和 maxlocks，以调整锁内存大小。另一个对数据库性能有较大影响的是排序内存，排序内存可以用于排序、HASH JOIN、MERGE JOIN、索引 AND 等操作。STMM 进程可以动态设置 sheapthres_shr 和 sortheap 的大小以适应内存大小需求变化。STMM 进程还可以动态调整程序包缓存大小，如图 1-18 所示。

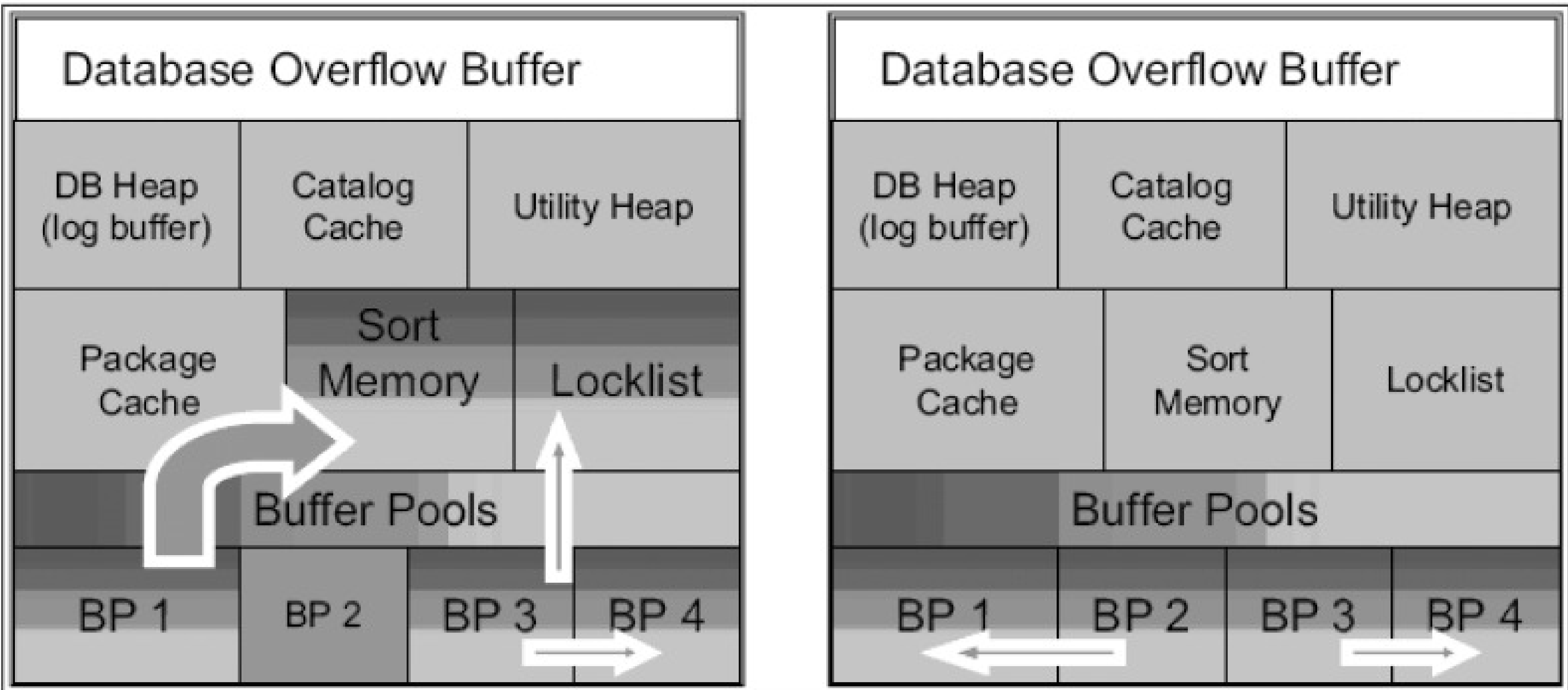


图 1-18　STMM 调优内存图

DB2 STMM 非常灵活，可同时支持 DBA 和 STMM 对可动态调整内存部分进行配置。在图 1-18 左边的例子中，STMM 动态调整排序内存、锁内存和大部分缓冲池。程序包缓存和 BP2 由 DBA 手工设置成具体值后，STMM 则不再动态调整这两个部分。在图 1-18 右边的例子中，STMM 仅仅对缓冲池进行动态调整，而锁内存、排序内存、程序包缓存则由 DBA 手工分配。

STMM 在两种模式下工作：

- 如果 DATABASE_MEMORY 被设置为 automatic，那么 STMM 可以从操作系统中获取内存或释放内存给操作系统，这允许数据库共享内存基于工作负载需求进行增长或收缩。在这种模式下，任何可动态调整的堆都可以被设置为 automatic，运行这些堆与操作系统动态交换内存。
- 如果 DATABASE_MEMORY 没有被设置成自动，那么数据库共享内存大小固定。在这种模式下，STMM 仅仅运行在各动态调整内存堆之间交换内存。这样至少需要两个堆被设置成 automatic。STMM 可以在缓冲池之间或者在缓冲池与排序内存之间交换内存，动态调整它们的大小。

关于内存自动调优示例及相关参数调优，本章不再赘述。

1.7　内存案例分析

由于数据库内存的分配和使用涉及实例和数据库多个不同的参数，下面通过具体示例就内存的使用给出一些经验总结。

假设当前 INSTANCE_MEMORY 为固定值 262144，实例启动后内存的分配如下：

```
    $db2pd -dbptnmem
    Database Partition 0 -- Active -- Up 0 days 07:22:13 -- Date
2012-11-23-17.57.04.835268
    Database Partition Memory Controller Statistics
    Controller Automatic: N
    Memory Limit:         1048576 KB(此值是 INSTANCE_MEMORY*4KB 大小)
    Current usage:         81152 KB  (此值是如下各 Memory Consumers 的 mem used 之和)
    HWM usage:           81152 KB  (此值是如下各 Memory Consumers 的 HWM used 之和)
    Cached memory:         15360 KB  (此值是如下各 Memory Consumers 的 Cached 之和)

    Individual Memory Consumers:
    Name            Mem Used (KB) HWM Used (KB) Cached (KB)
    ========================================================
    DBMS-caoww          55424        55424       15168
    FMP RESOURCES       22528        22528           0
    PRIVATE              3200         3200         192
```

当把 INSTANCE_MEMORY 的值改小到 100000 后内存的分配如下，可见如下各内存的使用者也都会随着 INSTANCE_MEMORY 的改变自动作出相应的改变，但各个值的计算方法不变。

```
    $db2 update dbm cfg using instance memory 100000
    $db2pd -dbptnmem
    Database Partition 0 -- Active -- Up 0 days 00:09:39 -- Date
2013-01-25-10.33.30.482454
    Database Partition Memory Controller Statistics
    Controller Automatic: N
    Memory Limit:         400000 KB  (此值是 INSTANCE_MEMORY*4KB 大小)
    Current usage:         82304 KB  (此值是如下各 Memory Consumers 的 mem used 之和)
    HWM usage:            82304 KB  (此值是如下各 Memory Consumers 的 HWM used 之和)
    Cached memory:         16064 KB  (此值是如下各 Memory Consumers 的 Cached 之和)

    Individual Memory Consumers:
    Name               Mem Used (KB) HWM Used (KB) Cached (KB)
    ========================================================
    DBMS-caoww           55424         55424        15168
    FMP RESOURCES        22528         22528            0
    PRIVATE               4224          4224          896
    LCL-p61407402          128           128            0
```

如果实例内存继续修改到小于 Current usage 的值，就会报出 SQL1362W 告警提示，当前的内存值在重启实例前不会发生任何变化。当重启后就会发现由于实例内存太小而启动不起来，此时又会报出 SQL1220N 错误。

```
    $db2 update dbm cfg using instance memory 1000
    SQL1362W  One or more of the parameters submitted for immediate modification
were not changed dynamically. Client changes will not be effective until the
next time the application is started or the TERMINATE command has been issued.
Server changes will not be effective until the next DB2START command.

    $db2stop force
    $db2start
    SQL1220N  The database manager shared memory set cannot be allocated.
    SQL1032N  No start database manager command was issued.  SQLSTATE=57019
```

通过上面示例的结果可知，实例内存不要调节到小于 Current usage 的值，强烈建议不要小于 HWM usage 的值。

在实际的运维过程中，为了减少数据库动态回收与分配内存的开销，强烈建议把实例内存设置成固定值。如果数据库服务器是在单独的机器上，建议把总物理内存的 70%分配给实例内存；如果和其他应用共享一台机器，建议把总物理内存的 30%分配给实例内存；如果实际的物理内存有限，那么根据实际情况，实例内存至少要达到 DB2 的 KPI 值。

由于各内存使用者的值没有统一的标准，下面给出实际运维中的经验设置值，后续的

调优工作可以根据此设置来进行。

实例参数设置推荐值：

```
AGENT STACK SZ >= 2048
ASLHEAPSZ = 15
RQRIOBLK = 65000
MON HEAP SZ = AUTOMATIC
JAVA HEAP SZ = 2048
AUDIT_BUF_SZ >= 0
```

数据库参数设置推荐值：

```
SELF TUNING MEM = OFF
DATABASE MEMORY = COMPUTED
DB MEM THRESH = 100
LOCKLIST >= 40000
MAXLOCKS = 90
PCKCACHESZ >= 7000
SHEAPTHRES SHR >= 20000
SORTHEAP >= 2048
DBHEAP = AUTOMATIC
CATALOGCACHE SZ >= 8000
LOGBUFSZ >= 1024
UTIL HEAP SZ >= 50000
BUFFPAGE >= 10000
STMTHEAP = AUTOMATIC
APPLHEAPSZ = AUTOMATIC
APPL MEMORY = AUTOMATIC
STAT_HEAP_SZ = AUTOMATIC
```

1.8　DB2 存储内部结构

在 DB2 数据库中，我们最终的用户数据是存放在存储上。了解 DB2 的存储内部结构对于我们合理的数据库物理设计和逻辑设计非常重要。

1.8.1　DB2 存储层次结构

在 DB2 系统中，数据存储被定义为 4 个层次，如图 1-19 所示。其中，分区组(Partition Group)是在多分区数据库中存在的一个层次。一个表空间包含一个或多个容器，但至少一个。在 DB2 中，容器可以是操作系统目录、预定义大小的文件或是未格式化的裸设备(如硬盘、Windows 磁盘分区、AIX 中的逻辑卷等)。

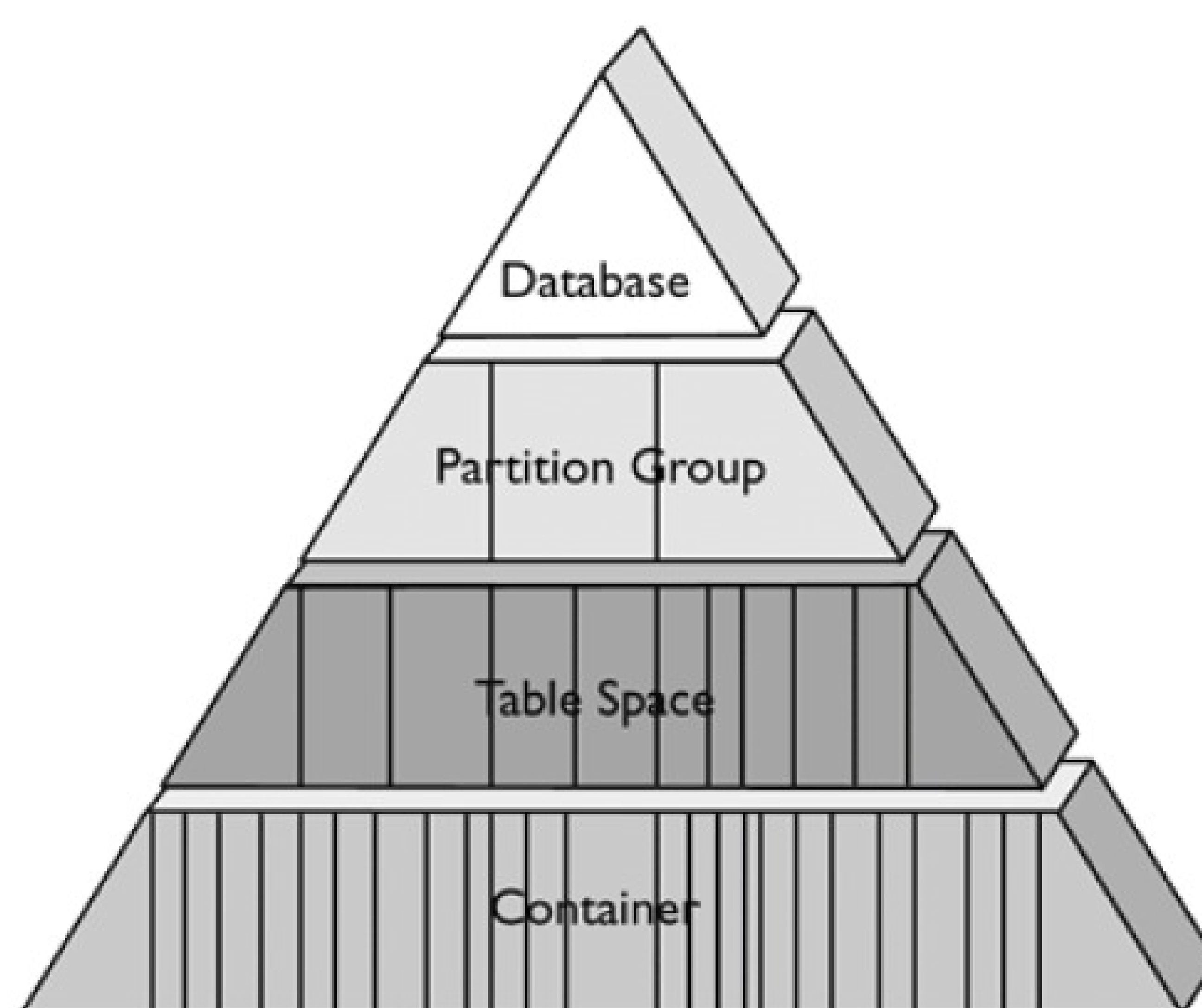


图 1-19　DB2 的存储层次结构图

用户创建的表、索引等数据库对象存放在表空间中。表直接面向应用，而同时表空间又和底层的物理存储对应，表空间可以有多个容器，而容器是在底层存储上的。所以通过表空间数据库实现了物理存储和逻辑存储的统一。

下面我们先讲解数据库的物理存储。我们都知道操作系统的最小存储单位是块(block)，在 Linux 和 UNIX 上，最小的块是 512 字节；在 Windows 上，最小的存储单位为 1KB。而数据库中最小的存储单位是数据页(data page)，它是 DB2 读写的最小单位。DB2 数据库中有 4KB、8KB、16KB 和 32KB 几种数据页。我们可以根据业务类型(OLAP、OLTP 等)和表的大小来选择合适的数据页。DB2 数据库在写物理存储时，为了保证写的吞吐量，引入了更大的单位 extent(扩展数据块)，它是整数倍的 data page 的大小。我们可以通过在创建表空间时指定 extentsize 来确定，而表空间容器又是由很多个 extent 组成的。它们之间的关系如图 1-20 所示。

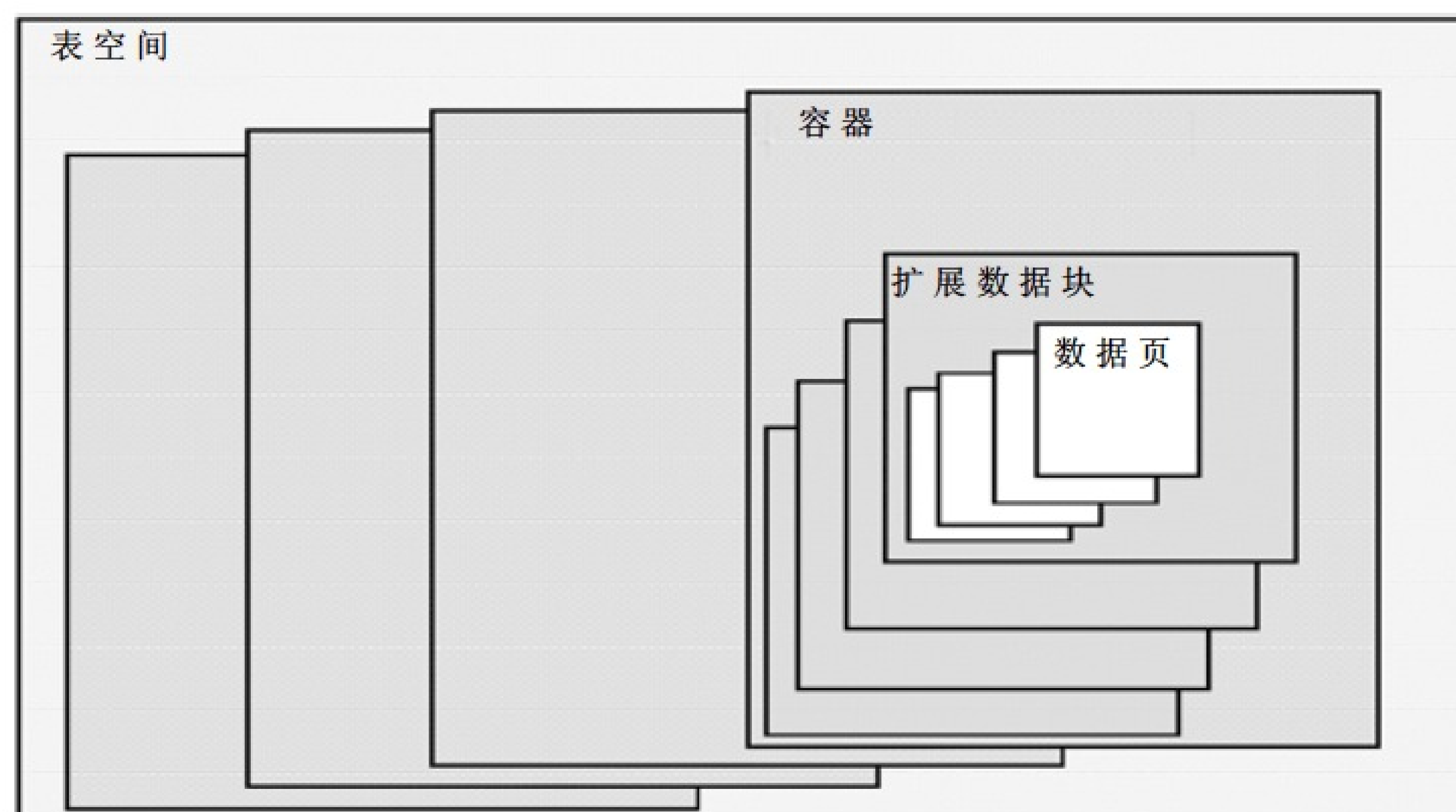


图 1-20　表空间、容器、extent 和数据页之间的关系

从前面的图 1-20 中我们知道，我们创建的表最终是存储在底层表空间容器上的，那么 DB2 如何来写容器呢？请看图 1-21。在图 1-21 中，第 1 个扩展数据块(Extent 0)分配在容器 0 上，第 2 个分配在容器 1 上，第 3 个分配在容器 2 上。此时每个容器都写完一个扩展数据块后，DB2 又返回到第一个容器以增加下一个扩展数据块，因此 Extent 3 在容器 0 上，Extent 4 分配在容器 1 上，依此类推，表空间的各个容器中均衡地存放了数据。

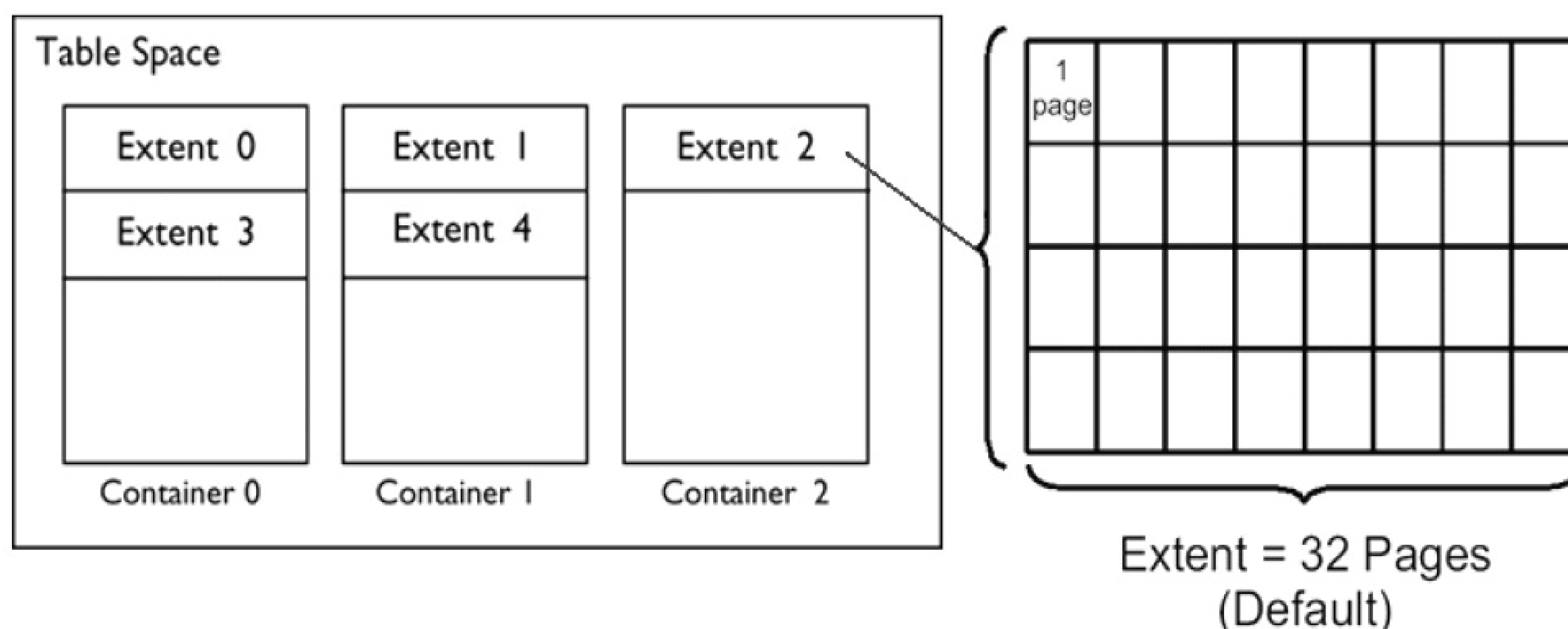


图 1-21　表空间容器被均衡写的示意图

1.8.2　表空间存储结构

DB2 支持三种管理类型表空间：

- **系统管理的空间(SMS)：**在这里，由操作系统的文件系统管理器分配和管理空间。
- **数据库管理的空间(DMS)：**在这里，由 DB2 数据库管理程序控制存储空间。表空间容器可使用文件系统或裸设备。
- **DMS 的自动存储(automatic storage with DMS)：**自动存储实际上不是一种单独的表空间类型，而是一种处理 DMS 存储的不同方式。

1.8.3　SMS 表空间的存储结构

SMS 表空间的容器采用操作系统文件系统的目录。在 SMS 表空间中，每一个表对应一个 DAT 文件，一个表的多个索引对应一个 INX 文件，大对象数据对应 LB 和 LBA 文件，long varchar 和 long vargraphic 对应 LF 文件。这就是 SMS 表空间的存储结构，一定要从操作系统级别注意维护好这些文件的权限和属组。

1.8.4　DMS 表空间的头部信息

要了解这个，就需要我们了解 DMS 表空间的头部信息，请看图 1-22，从这张图中我

们可以看到创建 DMS 表空间的最小大小是 6 个扩展数据块。试图创建小于 6 个扩展数据块的表空间将产生错误(SQL1422N)。这 6 个扩展数据块的用途如下：

- 第 1 个扩展数据块供表空间 TAG 使用。
- 表空间中有 3 个扩展数据块用于保存表空间元数据。
- 要存储任何用户表数据，至少需要两个扩展数据块(这些扩展数据块是表的常规数据所必需的，但不是任何索引、长字段或大对象数据所需的，它们需要自己的扩展数据块)。

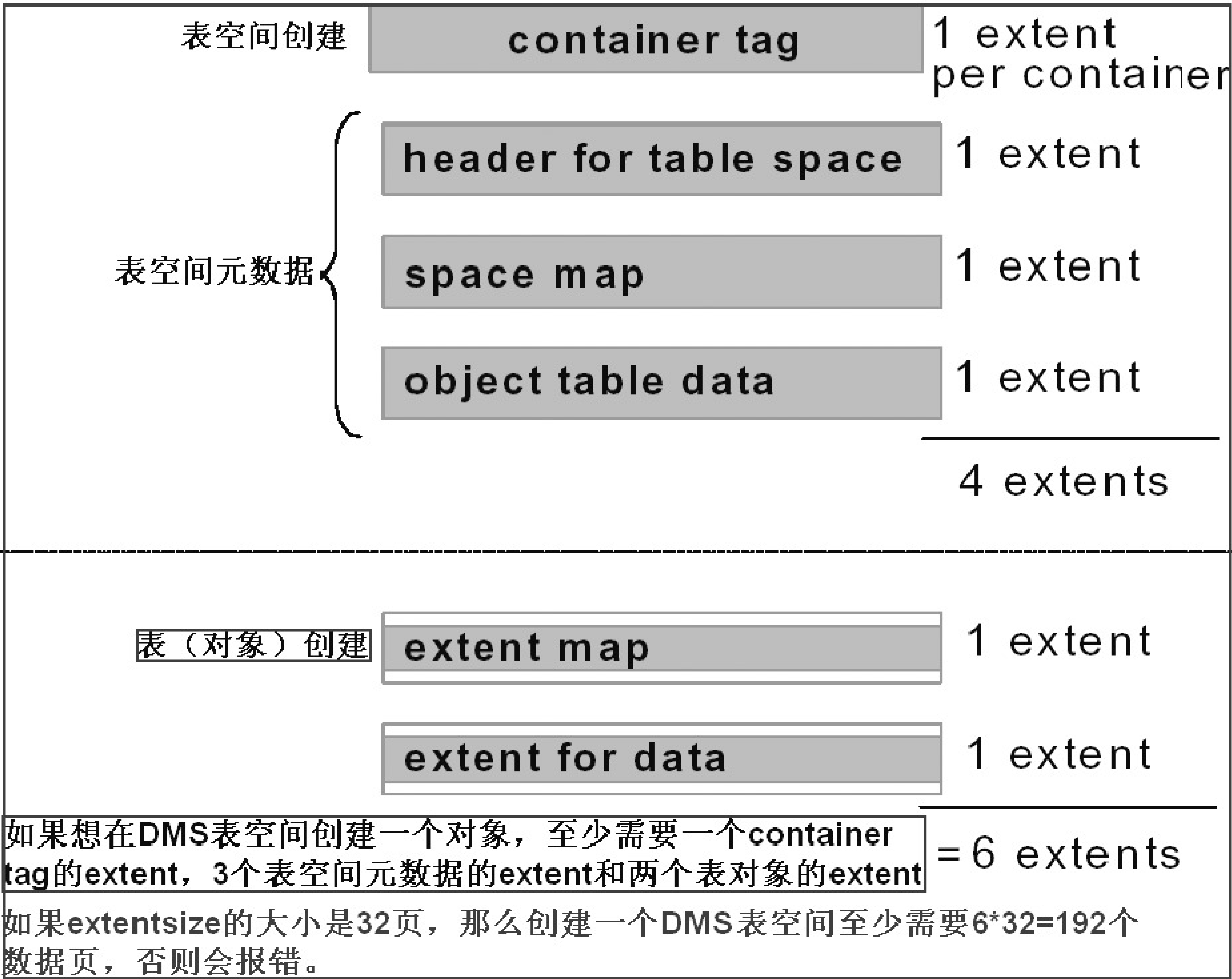


图 1-22　DMS 表空间的头部信息

1.8.5　DMS 表空间映射

表空间映射是数据库管理器的 DMS 表空间的内部表示，用于描述表空间中页位置的逻辑至物理转换。逻辑表空间地址映射中的扩展数据块以循环顺序在与表空间相关联的容器上进行条带分割。图 1-23 显示了 DMS 表空间的逻辑地址映射。

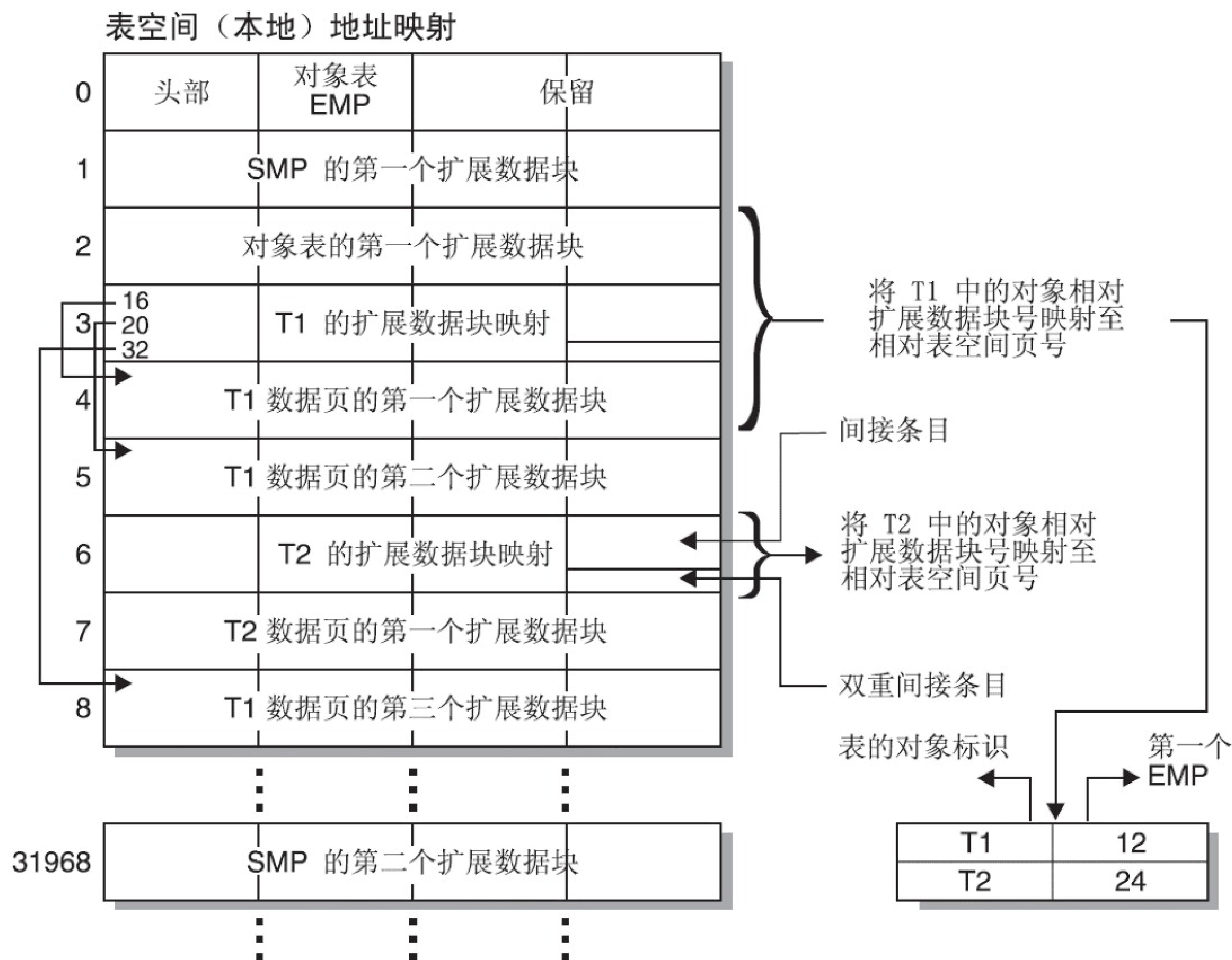


图 1-23　DMS 表空间的逻辑地址映射

1.8.6　表空间的高水位标记

高水位标记是表空间中分配的最高页的页数。例如，表空间有 1000 页，扩展数据块大小为 10，则结果为 100 个扩展数据块。如果第 42 个扩展数据块是表空间中最高分配的扩展数据块，那么意味着高水位标记是 42 * 10 =420 页。这与已使用的页不同，因为可能已经释放了高水位标记下的一些扩展数据块，所以它们可供复用。

1.8.7　RID 格式

记录标识(RID)由 4 个字节的页号及随后的 2 个字节的槽号组成。RID 的格式如图 1-24 所示。

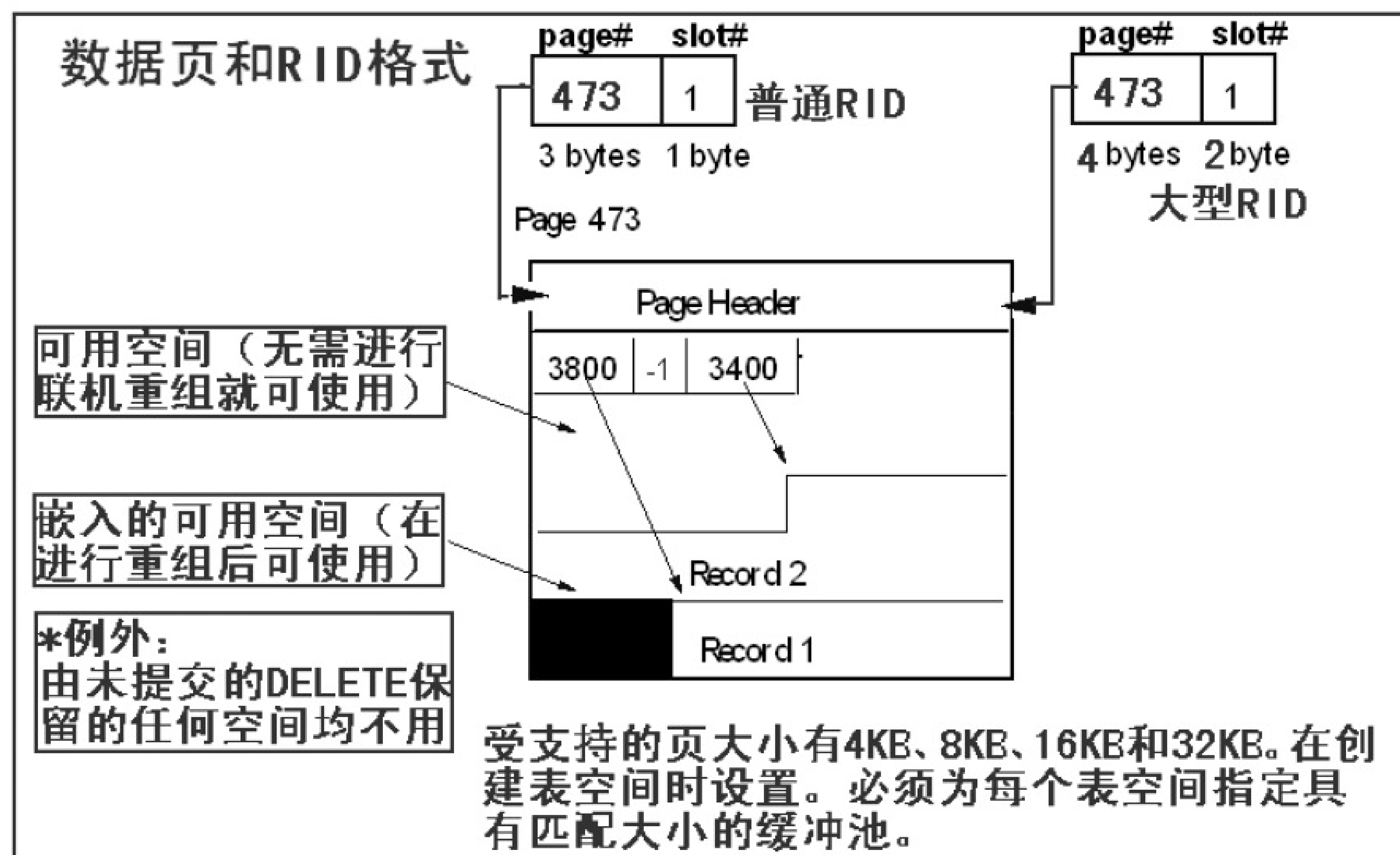


图 1-24　数据页和记录标识(RID)格式

在重组表时，实际删除记录后在页上留下的嵌入可用空间被转换成可使用的可用空间。根据记录在数据页上的移动重新定义 RID，以利用可使用的可用空间。

1.8.8　索引叶的内部结构

在 DB2 数据库中，索引是 B+树结构，索引的叶子如图 1-25 所示。

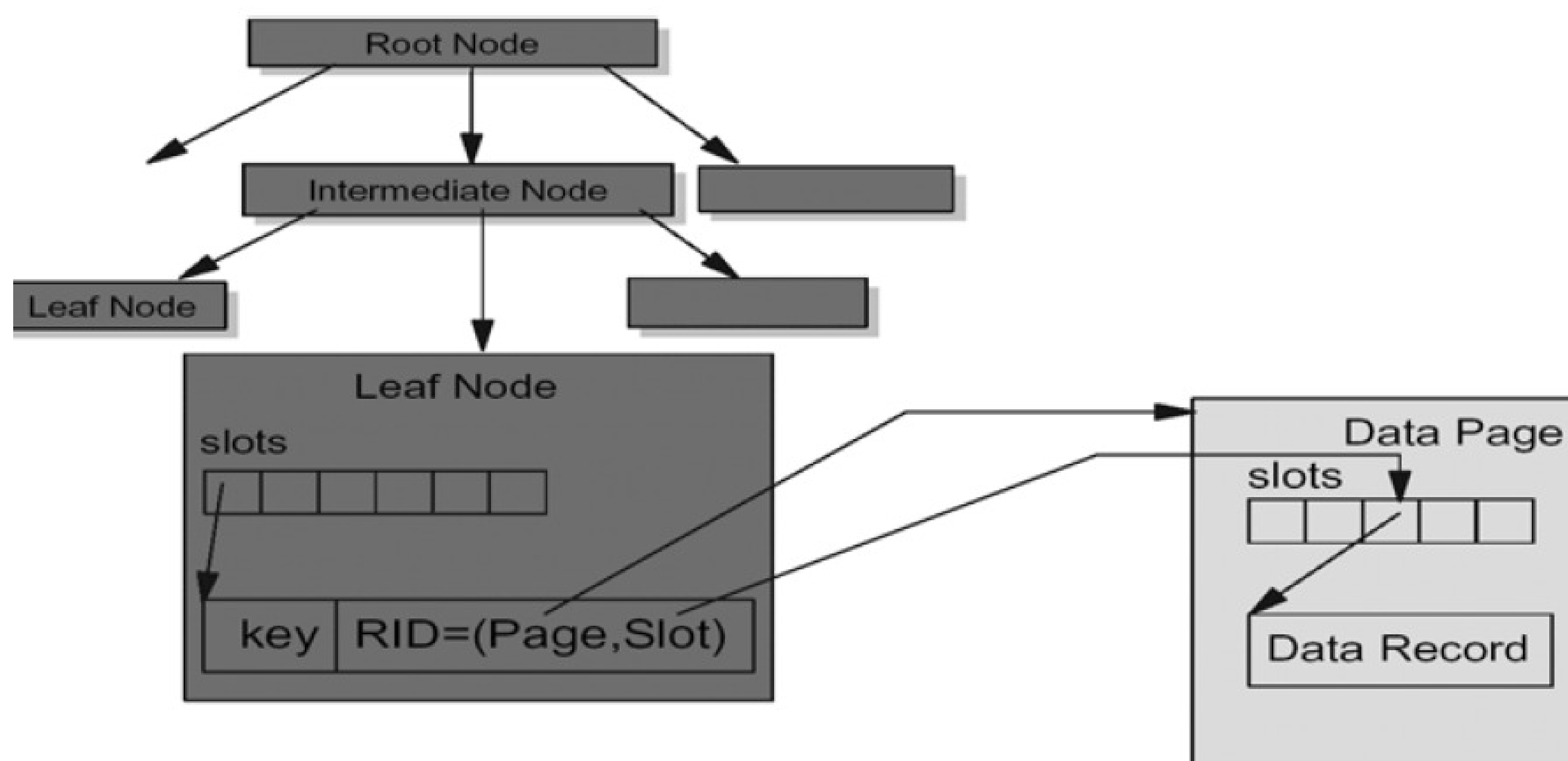


图 1-25　索引 B+树结构示意图

1.9　数据库物理设计

如果我们在进行存储 I/O 设计时已经在底层存储层面把物理存储设计好了，那么在数据库层面要做的物理设计工作就相对比较轻松和容易了。

1.9.1　表空间容器的放置原则

1. 容器设置

对操作系统逻辑卷和 RAID 存储级别实施条带化，建议表空间容器采用裸设备方式或支持并发 I/O 方式的文件系统。

2. 磁盘阵列和存储子系统

建议日志使用 RAID 5 级别，存放数据库数据的表空间容器采用 RAID 0+1 级别。

3. RAID 设备的条带化大小和 extentsize 大小保持一致

extentsize 应该等于物理存储 RAID 条带大小或其整数倍。prefetchsize 应该等于 RAID 条带大小乘以 RAID 并行设备数(或者等于该乘积的倍数)，而且这个值还应该是 extentsize 的倍数。建议在存储条带化大小、存储的 cachesize 大小、操作系统逻辑卷(LV)条带化大小和 extentsize 之间能够合理地配合设置。在 DB2 层面，DB2 提供了自己的注册变量 DB2_PARALLEL_IO，允许您增强特定环境。通过执行下面这个命令，可以在容器中启用 I/O 并行性：

```
db2set DB2_PARALLEL_IO=*
```

1.9.2　数据库物理设计原则

如果我们已经很好地设计了底层的物理存储，那么数据库的物理设计只需要把握以下原则。对于中等规模数据库及大规模数据库，应当考虑：

- 尽可能把应用的数据表空间、索引表空间以及相应的分区表空间分布在独立的 RAID GROUP 上。
- 其次是把存放数据的容器和数据库日志存放到不同的物理卷(RAID GROUP)上；
- 表空间容器使用裸设备，如果使用文件系统，建议使用 JFS2 类型的文件系统。
- 数据库日志文件都是顺序读写，建议采用 RAID 5 级别。
- 大部分 OLTP 数据库容器都是随机读写，建议采用 RAID 0+1 级别。对于 OLAP 建议采用 RAID 5。

1.10 数据库逻辑设计

1.10.1 缓冲池设计原则

数据库中的数据访问都需要经过缓冲池：读的数据需要先读到缓冲池才能提交给应用，写的数据也是要先写到缓冲池才能进行 I/O。缓冲池是影响数据库性能最大的参数，所以必须合理地设计缓冲池。

1. 缓冲池的重要性

数据库缓冲池是 DB2 环境中影响性能的最关键资源。DB2 的很多架构和设计，其基本思想都是尽可能地避免物理 I/O。

DB2 缓冲池由连续的内存页组成。数据和索引页被从物理存储容器中读出之后，便进入这些内存页中，并留在其中，直到 DB2 缓冲池管理器确定那些数据页要用于其他数据。应用程序请求的数据出现在内存中的概率越大，总体性能就越好。实际上，缓冲池里的数据被重复使用，因而减少了应用程序对 I/O 的需要。是否释放缓冲池脏页，这是根据最近被使用(LRU)原则来决定的。

创建数据库时，DB2 默认会自动创建名为 IBMDEFAULTBP 的缓冲池，所有的表空间都共享该缓冲池。可以使用 CREATE BUFFERPOOL 语句添加更多的缓冲池。缓冲池的默认大小由数据库配置参数 BUFFPAGE 指定，但是也可以通过在 CREATE BUFFERPOOL 命令中指定 SIZE 关键字来覆盖该值。足够的缓冲池大小是数据库拥有良好性能的关键所在，因为可以减少磁盘 I/O 这一最耗时的操作。大型缓冲池还会对查询优化产生影响，因为更多的工作可在内存中完成，而无须进行 I/O。

2. 基于块的缓冲池

DB2 允许您留出缓冲池的一部分(最高可达 98%)用于基于块的预取操作。基于块的 I/O 可以通过将块读入相邻的内存区而不是将块分散装入单独的页，进而提高预取操作的效率。每个缓冲池的块大小必须相同，并且由 BLOCKSIZE 参数进行控制。该参数的值等于块的大小(单位为页)，取值范围从 2 到 256，默认为 32。

3. CREATE BUFFERPOOL 语句示例

下面是 CREATE BUFFERPOOL 语句的一个示例：

```
CREATE BUFFERPOOL BP3 SIZE 2000 PAGESIZE 8K
```

提示：
基于块的缓冲池主要用于数据仓库、DSS 之类的连续大块读写的应用中。

在创建表空间之前，必须创建页大小和表空间一样的缓冲池。可以使用 ALTER TABLESPACE 命令将缓冲池添加到现有的表空间：

```
ALTER TABLESPACE USERSPACE3 BUFFERPOOL BP3
```

4. 查看缓冲池属性

通过查询 SYSCAT.BUFFERPOOLS 系统视图可以列出缓冲池信息：

```
SELECT * FROM SYSCAT.BUFFERPOOLS
BPNAME             BUFFERPOOLID NGNAME     NPAGES    PAGESIZE   ES
------------------ ------------ ------------------ ------------- --
IBMDEFAULTBP                  1 -                  250        4096 N
 1 record(s) selected.
```

要找出哪个缓冲池被分配给了表空间，请运行下面这个查询：

```
SELECT TBSPACE, BUFFERPOOLID FROM SYSCAT.TABLESPACES
TBSPACE            BUFFERPOOLID
------------------ ------------
SYSCATSPACE                   1
TEMPSPACE1                    1
USERSPACE1                    1
  3 record(s) selected.
```

可以在上面的查询中找到 BUFFERPOOLID，该查询使您能够看到每个表空间与哪个缓冲池相关联。

5. 数据库和表空间关系图

图 1-26 显示了数据库中表空间和缓冲池之间的关系。

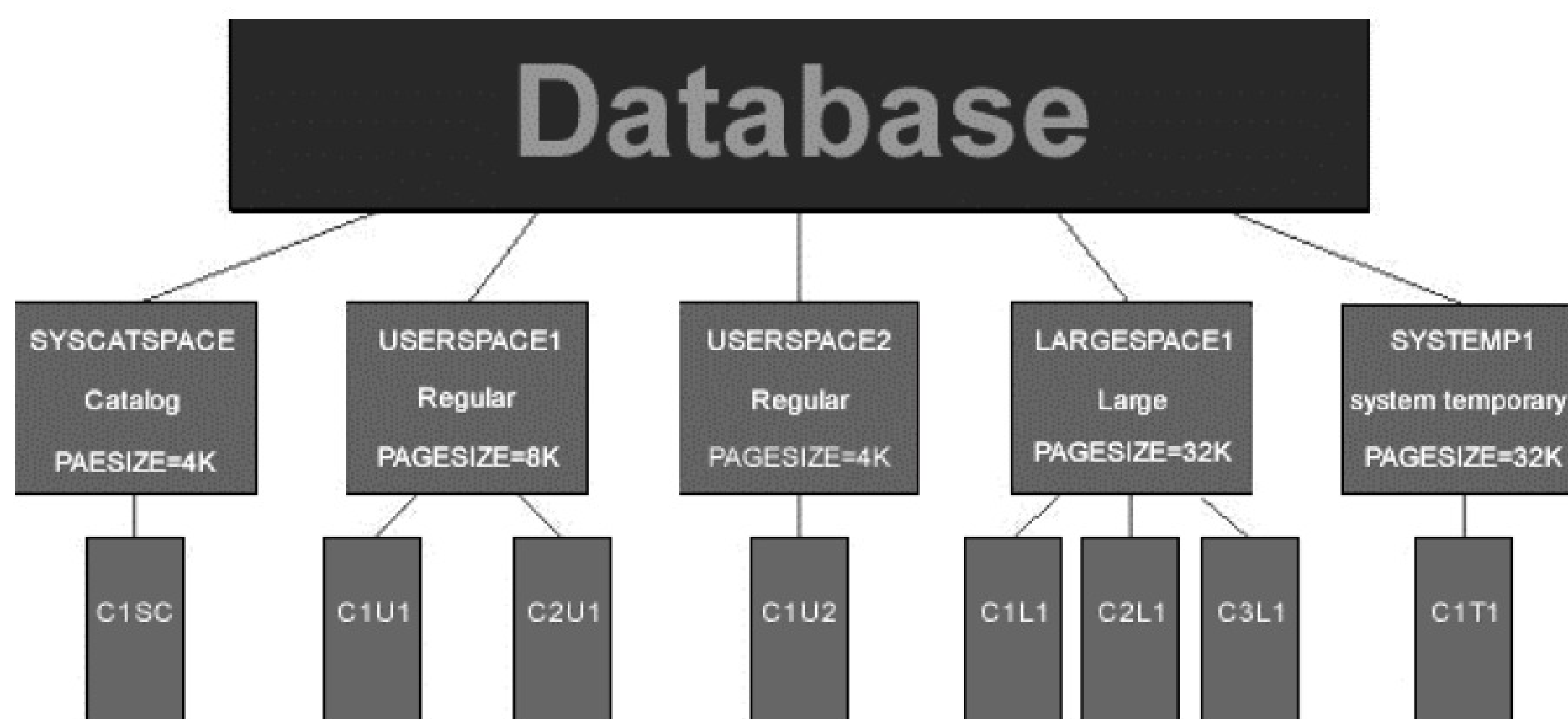


图 1-26　表空间和缓冲池之间的关系

该数据库有 5 个表空间：一个目录表空间、两个常规表空间、一个长表空间和一个系统临时表空间。没有创建用户临时表空间。另外有 8 个容器。

在这个方案中，缓冲池可能如下分配：

- 将 BP1(4KB)分配给 SYSCATSPACE 和 USERSPACE2
- 将 BP2(8KB)分配给 USERSPACE1
- 将 BP3(32KB)分配给 LARGESPACE1 和 SYSTEMP1

6. 缓冲池的利用率

使用多个用户表空间的最重要原因是管理缓冲池的利用率。一个表空间只能与一个缓冲池相关联，而一个缓冲池则可用于多个表空间。

缓冲池调优的目标是帮助 DB2 尽可能好地利用可用于缓冲池的内存。整个缓冲池大小对 DB2 性能有巨大影响，这是因为大的缓冲池可以显著地减少 I/O 这一最耗时的操作。但是，如果总的缓冲池设置太大，并且没有足够的物理内存来分配给它们，那么系统将会使用每种页大小最少的缓冲池，性能就会急剧下降。要计算最大缓冲池的大小，需要综合考虑 DB2、操作系统以及其他任何应用程序内存的使用率。一旦确定 DB2 总的可用内存大小，就可以将这个区域划分成不同的缓冲池以提高利用率。如果有一些具有不同页大小的表空间，那么每种页大小必须至少有一个缓冲池。

拥有多个缓冲池可以最大限度地将数据保存在缓冲池中。例如，假设数据库有许多频繁使用的小表，这些表通常全部都位于缓冲池中，因此访问起来就非常快。现在假设有一个针对非常大的表运行的查询，使用同一个缓冲池并且需要读取比总的缓存池大小还多的页。当查询运行时，之前来自这些频繁使用的小表的数据页将会作为“脏页”写到硬盘上，这使得再次需要这些数据时就必须重新读取它们。

如果小表拥有自己的缓冲池，那么它们就必须拥有自己的表空间，在这种情况下其他的查询就不能覆盖它们的数据页。这有可能产生更好的整体系统性能，虽然这会对大型查询造成一些小的负面影响。经常性地进行调优是为了实现整体的性能提高，而且时常需要在不同的系统功能之间做出权衡。区分功能的优先级并记住总吞吐量和使用情况，同时对系统性能进行调整，这是非常重要的。

DB2 V8 以后引入的新功能能够在不关闭数据库的情况下更改缓冲池大小。带有 IMMEDIATE 选项的 ALTER BUFFERPOOL 语句会立刻生效，只要数据库共享的内存中有足够的保留空间可以分配给新空间。可以使用这个功能，根据使用过程中的周期变化(例如从白天的交互式使用转换到夜间的批处理工作)来调优数据库性能。

7. 创建缓冲池

恰当地定义缓冲池是拥有运行良好的系统的关键因素之一。对于 32 位操作系统，知道共享存储器的界限十分重要，因为这种界限将限制数据库的缓冲池(数据库的全局存储器)，使其不能超出以下界限(64 位系统没有这样的界限)：

- AIX：1.75GB
- Linux：1.75GB
- Sun：3.35GB
- HP-UX：大约 800MB
- Windows：2GB~3GB (在 Windows NT/2000 的 boot.ini 中使用的是'3GB' switch)

用下面的公式计算近似的数据库全局存储器的使用(共享存储器)：

```
    buffer pools + dbheap + util heap sz + pkgcachesz + aslheapsz + locklist
+ approx 10% overhead
```

8. 确定有多少缓冲池

对于数据库中表空间使用的每一种页面大小，都需要至少一个缓冲池。通常，默认的 IBMDEFAULTBP 缓冲池是留给系统编目的。为处理表空间的不同页面大小和行为，必须创建新的缓冲池。

建议为每种页面大小使用一个缓冲池，对于 OLAP/DSS 类型的工作负载更是如此。DB2 在其缓冲池的自我调优方面十分擅长，并且会将经常被访问的行放入内存，因此多数情况下对于每一种页的大小创建一个缓冲池就足够了(这也避免了管理多个缓冲池的复杂性)。

如果时间允许，并且需要进行改进，那么您可能希望使用多个缓冲池。思路是将访问最频繁的行放入缓冲池中。在那些随机访问或者很少访问的表之间共享缓冲池可能会给缓

冲池带来“污染”，因为有时候要为本来可能不会再去访问的行消耗空间，甚至可能将经常访问的行挤出到磁盘上。如果将索引保留在它们自己的缓冲池中，那么在索引使用频繁的时候(例如索引扫描)还可以显著地提高性能。

这与对表空间的讨论是紧密联系的，因为要根据表空间中表的行为来分配缓冲池。如果采用多缓冲池的方法，对于初学者来说使用 4 个缓冲池比较合适：

- 一个中等大小的缓冲池，用于临时表空间。
- 一个大型的缓冲池，用于索引表空间。
- 一个大型的缓冲池，用于那些包含经常要访问的表的表空间。
- 一个小型的缓冲池，用于那些包含访问不多的表、随机访问的表或顺序访问的表的表空间。

对于 DMS 只包含 LOB 数据的表空间，可以为其分配任何缓冲池，因为 LOB 不占用缓冲池空间。

9. 确定为缓冲池分配的内存

千万不要为缓冲池分配多于所能提供的内存，否则就会招致代价不菲的操作系统内存分页(memory paging)。通常来讲，如果没有进行监控，要想知道一开始为每个缓冲池分配多少内存是十分困难的。

对于 OLTP 类型的工作负载，一开始将 25%(仅为参考，实际大小请参考自己操作系统中的内存资源和运行在操作系统中的应用情况)的可用内存分配给缓冲池比较合适。

对于 OLAP/DSS，经验法则告诉我们，应该将 40%(仅为参考，实际大小请参考自己操作系统中的内存资源和运行在操作系统中的应用情况)的可用内存分配给缓冲池(假设只有一种页大小)，同时监控排序情况，并对 sortheap 做相应调整。

10. 使用基于块(block-based)的缓冲池

对于有连续读写频繁的 OLAP 查询，可以得益于基于块的缓冲池。默认情况下，所有缓冲池都是基于页的，这意味着预取操作将把磁盘上相邻的页放入不相邻的内存中。而如果采用基于块的缓冲池，DB2 将使用块 I/O 一次将多个页读入缓冲池中，这样可以显著提高顺序预取的性能。

基于块的缓冲池由标准页区和块区同时组成。CREATE 和 ALTER BUFFERPOOL SQL 语句的 NUMBLOCKPAGES 参数用于定义块内存的大小，而 BLOCKSIZE 参数则指定每个块的大小，也就是在一次块 I/O 中从磁盘读取的页的数量。

共享相同区段大小的表空间应该成为特定的基于块的缓冲池的专门用户。将块大小设置为等于正在使用该缓冲池的表空间的扩展数据块大小。

确定分配多少内存给缓冲池内的块区要更为复杂一些。如果碰到大量的顺序预取操作，那么您很可能会想要更多基于块的缓冲池。NUMBLOCKPAGES 应该是块大小的倍数，并且不能大于缓冲池页面数量的 98%。先将它设小一点(不大于缓冲池总共大小的 15%或刚好 15%)。在后面还可以根据快照监视(snapshot monitoring)对其进行调整。

1.10.2　表空间设计原则

1. 表空间管理方式

可以用两种不同的方式管理表空间。

1) 系统管理的空间(SMS)

SMS 表空间由操作系统进行管理。容器被定义成常规操作系统文件，并且是通过操作系统调用访问的。这意味着所有的常规操作系统功能将处理以下内容：操作系统将缓冲 I/O；根据操作系统约定分配空间；如有必要就自动扩展表空间。但是，不能从 SMS 表空间删除容器，并且仅限于将新的容器添加到分区数据库。数据库默认创建的 3 个表空间都是 SMS 类型的表空间。

2) 数据库管理的空间(DMS)

DMS 表空间是由 DB2 管理的。可以将容器定义成文件系统(在创建表空间时将把给定的大小全部分配给它们)或裸设备。分配方法是操作系统允许多少 I/O，DB2 就可以管理多少 I/O。可以通过使用 ALTER TABLESPACE 命令来扩展容器。还可以释放未使用的那部分 DMS 容器。

下面是一个示例，向您说明该如何增大容器大小：

```
ALTER TABLESPACE TS1
RESIZE (FILE '/conts/cont0' 2000,DEVICE '/dev/rcont1' 2000,FILE 'cont2' 2000)
```

2. 表空间类型

数据库中的所有数据都存储在许多表空间中。表空间是表的容器，可以认为表空间是孩子而数据库是其父母。其中，表空间(孩子)不能有多个数据库(父母)。由于表空间有不同用途，因此根据它们的用途和管理方式将它们分类。根据用途有 5 种不同的表空间。

1) 系统目录表空间

每个数据库只有一个系统目录表空间，它是在发出 CREATE DATABASE 命令时创建的。目录表空间被 DB2 命名为 SYSCATSPACE(类似 Oracle 数据库的 SYSTEM 表空间、Sybase 数据库的 master 库、Informix 数据库的 sysmaster 库)，它保存了系统目录表。总是

在创建数据库时创建该表空间。

2) 大型表空间

大型表空间保存表数据和索引，此外还可以保存诸如大对象(Large Object，LOB)之类的长数据。大型表空间只能是数据库管理的空间(Database Managed Space，DMS)，可以将表及其索引分别存放到单独的常规表空间中。我们将在后面讲解 DMS 和系统管理的空间(System Managed Space，SMS)之间的区别。每个数据库中必须至少有一个大型表空间。创建数据库时指定该表空间的默认名为 USERSPACE1(类似 Oracle 数据库的 user 表空间)。

3) 常规表空间

常规表空间用于保存表数据和索引以及存储长型或 LOB 表列，它们可以是 SMS 表空间或 DMS 表空间。常规表空间是可选的，默认情况下一个都不创建。

注意：
建议读者创建表空间时尽量创建大型表空间。

4) 系统临时表空间

系统临时表空间用于存储 SQL 操作(例如排序、重组表、创建索引和连接表)期间所需的内部临时数据。每个数据库必须至少有一个系统临时表空间。随数据库创建的系统临时表空间的默认名为 TEMPSPACE1。

5) 用户临时表空间

用户临时表空间存储已声明的全局临时表。创建数据库时不存在用户临时表空间。至少应当创建一个用户临时表空间以允许定义已声明的临时表。用户临时表空间是可选的，默认情况下一个都不创建。

3. 表空间设置

可以在创建表空间时给它们指定许多设置，也可以在稍后使用 ALTER TABLESPACE 语句时指定其设置。

1) 页大小(Page Size)

定义表空间使用的页大小。所支持的大小有 4KB、8KB、16KB 和 32KB。

在 DB2 V8 中，表和表空间大小都是有限制的，如表 1-5 所示。表和表空间的大小限制取决于页大小。用作指针的字节数是 3 个字节。因此，只有 2 的 24 次方个单位可供使用。

由此可得到 16 777 216 个页面。由于页内的页槽号占 1 个字节，因此可寻址的行数为 255 乘以 16 777 216。

表 1-5　DB2 V8 中取决于页大小的表空间限制

页　　数	页　大　小	表/表空间限制
16 777 216	4KB	64GB
16 777 216	8KB	128GB
16 777 216	16KB	256GB
16 777 216	32KB	512GB

在 DB2 V9 中，这些限制被放宽了。用于页寻址的字节数增加到 4 个字节，页槽号现在用两个字节表示。表 1-6 显示了 DB2 V9 中的表和表空间限制。

表 1-6　DB2 V9 中取决于页大小的表空间限制

页　　数	页　大　小	表/表空间限制
536 870 912	4KB	2TB
536 870 912	8KB	4TB
536 870 912	16KB	8TB
536 870 912	32KB	16TB

在 DB2 V9.7 和 V10.5 中，用于页寻址的字节数仍是 4 个字节，但表和表空间限制进一步放宽了，如表 1-7 所示。

表 1-7　DB2 V9.7 和 V10.5 中取决于页大小的表空间限制

页　　数	页　大　小	表/表空间限制
2147483648	4KB	8TB
2147483648	8KB	16TB
2147483648	16KB	32TB
2147483648	32KB	64TB

大型 RID 只在 DB2 V9 以上版本的大型表空间中受支持。这不同于 DB2 V8，在 DB2 V8 中，大型表空间只是为 LOB 和 LONG 数据类型设计的。但是当从 DB2 V8 迁移至 DB2 V9 时要记住，常规表空间不会被转换成大型表空间。在迁移计划中，要考虑到可能需要将常规表空间转变为大型表空间。

对于执行随机更新操作的 OLTP 应用程序，采用较小的页面大小更为可取，因为这样消耗的缓冲池中的空间更少。

对于要一次访问大量连续行的 OLAP 和 DSS 应用程序，通常使用较大页面大小效果会更好些，因为这样可以减少在读取特定数量的行时发出的 I/O 请求的数量。较大的页面大小还允许您减少索引中的层数，因为在一页中可以保留更多的行指针。然而，也有例外情况。如果行长度小于页面大小的 1/255，那么每一页中都将存在浪费的空间，因为每页最多只能有 255 行(对于索引数据页不适用)。在这种情况下，采用较小的页面大小或许更合适一些。

例如，对于常规表空间，如果要使用 32KB 的页面大小来存储平均大小为 100 字节的行，那么 32KB 的页只能存储 100 * 255 = 25500 Byte(24.9 KB)。这意味着每 32KB 中就有大约 7KB 要浪费掉。

2) 扩展块大小(extentsize)

指定在跳到下一个容器之前将写到当前容器中的页数。存储数据时数据库管理器反复循环使用所有容器。该参数只有在表空间中有多个容器时才起作用。

extentsize 指定在跳到下一个容器之前，可以写入到容器中的 pagesize 页面的数量，这个参数是在创建表空间时定义的(之后不能轻易修改)。处理较小的表时，使用较小的扩展数据块效率会更高一些。

下面的经验法则建立在表空间中每个表的平均大小的基础上：

- 如果小于 25MB，extentsize 为 8。
- 如果介于 25MB 到 250MB 之间，extentsize 为 16。
- 如果介于 250MB 到 2GB 之间，extentsize 为 32。
- 如果大于 2GB，extentsize 为 64。

对于 OLAP 数据库和大部分都要扫描(仅限于查询)的表，或者增长速度很快的表，应使用较大的值。

如果表空间驻留在磁盘阵列上，那么应将区段大小设置成条带化大小的整数倍(也就是说，写入到阵列中某个磁盘上的数据)。

3) 预取大小(prefetchsize)

当执行数据预取时每次从表空间读取的页数。预取操作在查询使用所需的数据之前读

入这些数据，因为数据已经存在于内存中了，这样一来，查询在使用这些数据的时候就不必等待执行 I/O 了。当数据库管理器确定顺序 I/O 是适当的，并且确定预取操作可能有助于提高性能时，就选择预取操作。

通过使用 ALTER TABLESPACE 可以轻易地修改预取大小。最优设置差不多是下面这样的：

```
    prefetchsize = (# Containers of the table space on different physical disks)
                   * Extent Size
```

如果表空间驻留在磁盘阵列上，那么设置如下：

```
    prefetchsize = extentsize * (# of non-parity disks in array)。
```

注意：

在 DB2 V9 以后版本中，可以在创建表空间的时候自动预取大小。

可以在创建表空间时指定 prefetchsize 为 automatic，这样就可以设置自动预取大小了，并可以通过下面的快照监控来查看是否设置了自动预取：

```
db2 get snapshot for tablespaces on sample | more
               Tablespace Snapshot

    First database connect timestamp          = 10/10/2012 10:38:38.725432
    Last reset timestamp                      =
    Snapshot timestamp                        = 10/11/2012 08:54:23.527872
    Database name                             = SAMPLE
    Database path                             =
/db2data/inst97/inst97/NODE0000/SQL00001/
    Input database alias                      = SAMPLE
    Number of accessed tablespaces            = 11

    Tablespace name                           = SYSCATSPACE
      Tablespace ID                           = 0
      Tablespace Type                         = Database managed space
      Tablespace Content Type                 = All permanent data. Regular table space.
      Tablespace Page size (bytes)            = 8192
      Tablespace Extent size (pages)          = 4
      Automatic Prefetch size enabled         = Yes
      Buffer pool ID currently in use         = 1
```

```
     Buffer pool ID next startup                = 1
     Using automatic storage                   = Yes
     Auto-resize enabled                      = Yes
     File system caching                      = No
     Tablespace State                        = 0x'00000000'
      Detailed explanation:
        Normal
     Tablespace Prefetch size (pages)           = 4
     Total number of pages                     = 16384
     Number of usable pages                    = 16380
     Number of used pages                      = 14388
     Number of pending free pages                = 0
     Number of free pages                      = 1992
     High water mark (pages)                    = 14388
     Initial tablespace size (bytes)             = 33554432
     Current tablespace size (bytes)             = 134217728
     Maximum tablespace size (bytes)             = NONE
     Increase size (bytes)                     = AUTOMATIC
     Time of last successful resize               =
     Last resize attempt failed                  = No
     Rebalancer Mode                         = No Rebalancing
     Storage paths have been dropped             = No
     Minimum Recovery Time                     =
     Number of quiescers                       = 0
     Number of containers                      = 1

     Container Name                          =
/db2data/inst97/inst97/NODE0000/SAMPLE/T0000000/C0000000.CAT
        Container ID                         = 0
        Container Type                       = File (extent sized tag)
        Total Pages in Container                = 16384
        Usable Pages in Container               = 16380
        Stripe Set                          = 0
        Container is accessible                 = Yes
        File system ID                       = 9223372079804448783
        File system used space (bytes)           = 2419458048
        File system total space (bytes)          = 21474836480
```

```
Table space map:

 Range  Stripe Stripe  Max         Max  Start  End    Adj.   Containers
 Number Set    Offset  Extent      Page Stripe Stripe
 [   0] [   0]      0    4094     16379      0   4094    0    1 (0)

Buffer pool data logical reads          = 28378
Buffer pool data physical reads         = 418
Buffer pool temporary data logical reads   = 0
Buffer pool temporary data physical reads  = 0
Asynchronous pool data page reads        = 33
Buffer pool data writes                = 51
Asynchronous pool data page writes       = 50
Buffer pool index logical reads          = 14271
Buffer pool index physical reads         = 576
Buffer pool temporary index logical reads  = 0
Buffer pool temporary index physical reads = 0
Asynchronous pool index page reads       = 6
Buffer pool index writes               = 40
Asynchronous pool index page writes      = 40
Buffer pool xda logical reads           = 0
Buffer pool xda physical reads          = 0
Buffer pool temporary xda logical reads  = 0
Buffer pool temporary xda physical reads = 0
Asynchronous pool xda page reads         = 0
Buffer pool xda writes                 = 0
Asynchronous pool xda page writes        = 0
Total buffer pool read time (millisec)    = 2317
Total buffer pool write time (millisec)  = 155
Total elapsed asynchronous read time     = 95
Total elapsed asynchronous write time    = 148
Asynchronous data read requests          = 13
Asynchronous index read requests         = 5
Asynchronous xda read requests          = 0
No victim buffers available             = 245
Direct reads                         = 32350
```

```
    Direct writes                              = 66
    Direct read requests                        = 2161
    Direct write requests                       = 8
    Direct reads elapsed time (ms)               = 3184
    Direct write elapsed time (ms)               = 86
    Number of files closed                      = 0
  略……………………
```

4) 开销(Overhead)和传送速率(Transrate)

这些值用于确定查询优化期间的 I/O 成本。这两个值的测量单位都是毫秒，而且它们应当分别是所有容器开销和传送速率的平均值。开销是与 I/O 控制器活动、磁盘寻道时间和旋转延迟时间相关联的时间。传送速率是将一页读入内存所需的时间量。它们的默认值分别是 24.1 和 0.9。可以根据硬件规格计算这些值。

Transrate=(1/传送速率)*1000/1024000*4096(假设用 4KB 页大小)

Overhead=平均寻道时间+(((1/磁盘转速)*60*1000)/2)

而平均寻道时间、磁盘旋转速度和传送速率是由硬盘本身决定的。

5) 容器

每个表空间都可以有一个或多个容器。重申一次，您可以认为容器是孩子，而表空间是其父母。每个容器只能属于一个表空间，但是一个表空间可以拥有许多容器。可以将容器添加到 DMS 表空间，或者从 DMS 表空间中删除容器，而且可以更改容器的大小。在分区数据库环境中，只能将容器添加到某个分区中数据库的 SMS 表空间，在添加之前该分区还未给表空间分配容器。添加新的容器时，将启动自动的重新均衡操作以便将数据分布到所有容器上。重新均衡操作不会妨碍对数据库的并发访问。

4. 表空间设计原则

1) 确定表空间的数量

与缓冲池一样，一开始应该为每种页大小使用一个缓冲池。对于使用的每种页大小，必须存在一个具有匹配页大小的系统临时表空间(以支持排序和重组)。然后将所有相同页大小的表空间指派给具有相同页大小的缓冲池。

如果您还关心性能问题，并且有时间投入，那么可以使用 DMS 表空间，并且根据使用情况来组织表。另外，还要遵循前面给出的关于使用多个缓冲池的建议。

对于系统临时表空间和系统编目表空间，应该使用 SMS，因为它允许表空间动态地增长和收缩。如果有大量临时表要刷新到磁盘上(或是没有足够的排序空间，或是显式地创建临时表)，那么 DMS 会更有效一些。

对于所有其他的表空间，应该使用 DMS。DMS 允许一个表跨越多个表空间(索引、用户数据和 LOB)，这样就减少了在预取和更新操作时索引、用户和 LOB 数据之间的争用，从而缩短了数据访问的时间。表空间的容器通过使用裸设备或支持并发 I/O 的文件系统可以得到额外 5%~10%的性能提升。

通常应该将系统目录表空间和系统临时表空间作为 SMS 分配。没有必要拥有多个具有相同页大小的临时表空间，通常只需一个具有最大页大小的临时表空间就够了。

突出的问题在于是否要将用户数据分割到多个表空间中。一个考虑因素是页的利用率。不能将行分割到不同的页，因此，具有长行的表需要有合适的页大小。但是，常规表空间数据页上的行不能超过 255 个，因此具有较短行的表不能利用整个页。例如，在页大小为 32KB 的表空间中放置行长度为 12 字节的表，大约只能利用每个页的 10%，即(255 行* 12 字节+ 91 字节的开销)/32KB 页大小≈10%。

如果表很大，这只是一个考虑因素，因此浪费的空间就非常大。此外还会使 I/O 和缓存的效率降低，因为每个页的实际有用内容很少。如果可以将表放到具有较小页大小的表空间中，并且可以充分利用较大的页大小，那么最常用的访问方法将确定哪一个更好。如果通常是顺序访问大量行(该表可能进行了群集)，那么比较大的页大小会比较有效。如果随机访问行，那么较小的页大小可以允许 DB2 更好地利用缓冲区，因为同样的存储区域可以容纳更多页。

一旦根据页大小对表进行了分组，那么访问频率和类型将确定把数据进一步分组到独立的表空间中是否有意义。每张表根据自己被最频繁访问的方式，可以具有一组最有效的表空间设置：pagesize、extentsize 和 prefetchsize。上面已介绍了 pagesize。extentsize 是在将数据写入到下一个容器之前，写入到当前容器中的数据的页数(如果表空间中存在多个容器的话)。

prefetchsize 指定在执行数据预取时将从表空间读取的页数。当数据库管理器确定顺序 i/o 是适当的，并且确定预取操作可能有助于提高性能时，会使用预取操作(通常是全表扫描)。比较好的做法是将 prefetchsize 的值显式地设置成表空间的 extentsize 值与表空间容器数的乘积的倍数。例如，如果 extentsize 是 32，并且表空间中有 4 个容器，那么理想的 prefetchsize 应当是 128、256 等等。如果一个或多个频繁使用的表需要的这组参数的值不同于那些最适用于表空间其他表的性能的参数值，那么将这些表放入单独的表空间可能会提高整体性能**(在 DB2 V9 版本以后，可以在创建表空间的时候自动预取大小。您可以在创建表空间时指定 prefetchsize 为 automatic，这样就可以设置自动预取大小了)。**

如果预取操作是表空间中的重要因素，那么请考虑留出一部分缓冲池用于基于块的 I/O。块大小应当等于 prefetchsize。

2) 自动存储

什么是自动存储？自动存储(automatic storage)是 DB2 V9 开始引入的新特性，它允许为数据库指定一个或多个存储路径。当您创建表空间时，DB2 自动将表空间放在指定的存储路径上。在创建数据库时，可以为之启用或配置自动存储，命令如下：

```
db2 create database db_name automatic storage yes
db2 create database db_name on db_path1, db_path2
```

还可以使用 add storage 参数为设置了自动存储的数据库添加附加的存储路径，方法如下：

```
db2 alter database db_name add storage on  db_path3
```

一旦为数据库设置了自动存储，就可以使用这种机制来创建表空间。为数据库设置了自动存储后，您可以有多种方法来利用自动存储。您可以在数据库中创建表空间(在连接到数据库之后)，如下所示：

```
db2 create tablespace ts_name
```

或者，也可以创建表空间并指定它的初始大小和增长特征，如下所示：

```
db2 create tablespace ts_name
     initialsize 10M
     increasesize 10M
     maxsize 100M
```

在上述代码中，表空间的初始大小为 10MB。当表空间接近大小限制时，DB2 每次自动将表空间扩大 10MB，直到表空间达到 100MB 的最大值。

如果数据库没有设置自动存储，那么可以创建表空间并指定它的存储大小，然后为之使用自动存储：

```
db2 create tablespace ts_name  managed by automatic storage
```

5. 表空间状态

要查看数据库中表空间的状态，可以使用命令：

```
list tablespaces show detail
```

表空间可以有多种不同的状态，如下所示：

0x0	Normal
0x1	Quiesced Share
0x2	Quiesced Update
0x4	Quiesced Exclusive
0x8	Load Pending
0x10	Quiesced Share
0x20	Delete Pending
0x40	Roll Forward in Progress
0x80	Roll Forward Pending
0x100	Restore Pending
0x200	Disable Pending
0x400	Reorg in Progress
0x800	Backup in Progress
0x1000	Storage Must be Defined
0x2000	Restore in Progress
0x4000	Offline and Not Accessible
0x8000	Drop Pending
0x20000	Load in Progress
0x2000000	Storage May be Defined
0x10000000	DMS Rebalance in Progress
0x20000000	Table Space Deletion in Progress
0x40000000	Table Space Creation in Progress

1.10.3　索引设计原则

虽然构成索引键的列的顺序不会影响索引键的创建，但是当决定是否使用索引时就可能影响优化器。例如，如果查询包含“ORDER BY col1,col2”子句，那么可以使用为(col1,col2)创建的索引，但为(col2,col1)创建的索引将没有作用。同样，如果查询指定了条件，例如“where col1 >= 50 and col1 <= 100”或“where col1=74”，那么为(col1)或(col1,col2)创建的索引将起作用。

注意：

每当有可能时，都应该按最独特到最不独特的顺序对索引键中的列进行排序。这将使性能最佳。

可以对特定的表定义任意数目的索引(最大数目为 32 767)，这些索引能提高查询性能。索引管理器必须在更新、删除和插入操作期间维护索引。为接收很多更新内容的表创建大量索引可能减慢请求的处理速度。同样，大型索引键也会减慢处理请求的速度。因此，仅当频繁访问有明显有利之处时，才使用索引。

不是唯一索引键的一部分，但要在索引中存储或维护的列数据称为包含列。只能为唯一索引指定包含列。当用包含列创建索引时，仅对唯一键列进行排序并考虑其唯一性。使用包含列可以启用仅访问索引来进行数据检索，从而提高性能。

如果要建立索引的表是空的，那么仍会创建索引，但是在装入表或插入行之前，不会建立任何索引条目。如果该表不为空，那么数据库管理器将在处理 CREATE INDEX 语句时创建索引条目。

对于集群索引，数据库管理器会尝试将表的新行插入到具有(由索引定义的)相似键值的现有行附近。

如果要让主键索引成为集群索引，那么不应在 CREATE TABLE 语句中指定主键。一旦创建主键，就不能修改相关的索引。而是发出不带主键的 CREATE TABLE。然后，发出 CREATE INDEX 语句并指定集群属性。最后，使用 ALTER TABLE 语句添加与刚创建的索引对应的主键。把此索引用作主键索引。

如果有分区表，那么在默认情况下，创建的任何索引都是分区索引，除非创建不包括分区键的唯一索引。还可以将索引作为非分区索引来创建。

从 DB2 V9.7 修订包 1 开始，可以对分区表创建基于 XML 数据的分区索引或非分区索引。默认情况下将创建分区索引。

对分区表执行滚入操作时(使用 ALTER TABLE 语句的 ATTACH PARTITION 子句将数据分区连接到另一个表)，分区索引具有优势。借助分区索引，可以避免必须对非分区索引执行的索引维护工作。如果分区表使用非分区索引，那么必须使用 SET INTEGRITY 语句对新组合的数据分区执行索引维护。此操作不仅耗时，而且可能需要大量的日志空间，这取决于正在滚入的行数。

索引会消耗磁盘空间。磁盘空间的大小取决于键列的长度和要建立索引的行数。随着插入到表中的数据增多，索引大小也会增加。因此，在规划数据库大小时，应考虑正在建立索引的数据量。下面是一些建立索引大小时的注意事项：

- 主键和唯一键约束始终创建系统生成的唯一索引。
- 创建 MDC 表时也将创建由系统生成的块索引。
- XML 列将始终导致创建由系统生成的索引(其中包括列路径索引和区域索引)。
- 对外键约束列创建索引通常会有好处。
- 是否使用 COMPRESS 选项对索引进行压缩。

注意：

索引中的最大列数为 64。但是，如果对类型表建立索引，那么索引中的最大列数为 63。索引键的最大长度(包括所有开销)为 IndexPageSize/4。表上允许的最大索引数为 32 767。索引键的最大长度不能大于页大小的索引键长度限制。

在数据库升级期间，不会对现有索引进行压缩。如果对表启用数据行压缩功能，那么除非在 CREATE INDEX 语句中指定 COMPRESS NO 选项，否则将对升级后创建的新索引进行压缩。

- 用于设计索引的工具

创建表后，需要考虑数据库管理器能够从这些表中检索数据的速度。可以使用 db2advis 命令来帮助您设计索引。

对表创建有用的索引可以极大地提高查询性能。与书籍的索引一样，使用表索引就可以通过最少的搜索而快速找到特定信息。使用索引从表中检索特定行可以减少数据库管理器需要执行的成本较高的输入/输出操作数。这是因为索引允许数据库管理器通过读取相对较少的数据页来找到行，而不是彻底搜索所有数据页直到找到所有匹配项为止。

DB2 设计顾问程序是可以帮助您显著提高工作负载性能的工具。选择要为复杂工作负载创建哪些索引、MQT、集群维度或数据库分区的任务可能会令人十分头痛。“设计顾问程序”标识了提高工作负载性能所需要的所有对象。如果工作负载中存在一组 SQL 语句，那么设计顾问程序将为下列各项提供一些建议：

- 新的索引
- 新的具体化查询表(MQT)
- 对多维集群(MDC)表的转换
- 表的重新分发
- (通过 GUI 工具)删除指定的工作负载未使用的索引和 MQT

1.11　本章小结

本章介绍了 DB2 的进程以及内存和存储的体系结构，详细讲解了 DB2 进程体系结构、代理程序通信机制及实用程序相关进程，并且详细介绍了数据库的底层存储结构和相关的物理设计和逻辑。了解 DB2 内部的进程体系结构可以帮助我们确定问题的性质，因为它会帮助我们理解数据库管理器、数据库、应用程序和实用程序的内部工作原理。同时内存是 DB2 从操作系统获取的最重要资源，内存对数据库的性能影响非常重要，我们介绍了 DB2 中内存集、内存池、内存块这三种内存的分配、回收机制。DB2 从操作系统中申请到的内存主要被实例、数据库、应用程序和代理进程所消耗，本章讲述了各个消耗单位与配置参数之间的关系。在过去，我们要小心谨慎地设置 DB2 中的实例、数据库和应用程序相关配置参数以保证合理的资源分配。随着 DB2 技术的发展，内存的设置将变得越来越方便，因此我们给大家讲解了 DB2 内存自动调优的概念，内存自动调优可以保证我们合理地利用内存资源，从而保证数据库高性能运转。考虑周详的数据库设计可以提供巨大的性能收益，

但是这必须在应用程序开发过程的早期便开始着手。从应用系统构建的早期开始，开发人员就应该使用我们前面讲到的准则。当数据库性能成为开发过程中的焦点时，良好的数据库设计使得为 DB2 应用程序提供最佳性能有了更大的可能性。

第 2 章

DB2 表的高级特性

在《循序渐进 DB2(第 3 版)》中，我们介绍了表的一些基本特性，比如 pctfree、locksize、volatile、append on 等，但是只使用这些属性来应付数据仓库级的数据量是远远不够的，因此本章将重点介绍几个在仓库级数据库里常用的特性：表分区、MDC、MQT 和行压缩。

2.1 表分区

2.1.1 定义

表分区功能是一种数据组织方案，换言之，表数据根据一个或多个表列中的值分布到多个存储对象(称为数据分区或范围)中。每个数据分区都是单独存储的。这些存储对象可以在不同的表空间中，也可以在相同的表空间中。

存储对象的行为与单个表的行为非常类似，通过使用 ALTER TABLE ... ATTACH 语句将现有表合并到分区表中，可以很容易实现快速转入。同样，使用 ALTER TABLE ... DETACH 语句很容易实现转出。查询处理同样可以利用分离的数据来避免扫描不相关数据，从而使许多数据仓库样式查询具有更好的查询性能。DB2 中的表分区特性如图 2-1 所示，最终用户看到的还是一张表，但这张表底层被分成了多个逻辑分区，每个分区又可以看成是一张独立的表，可以选择自己的表空间。

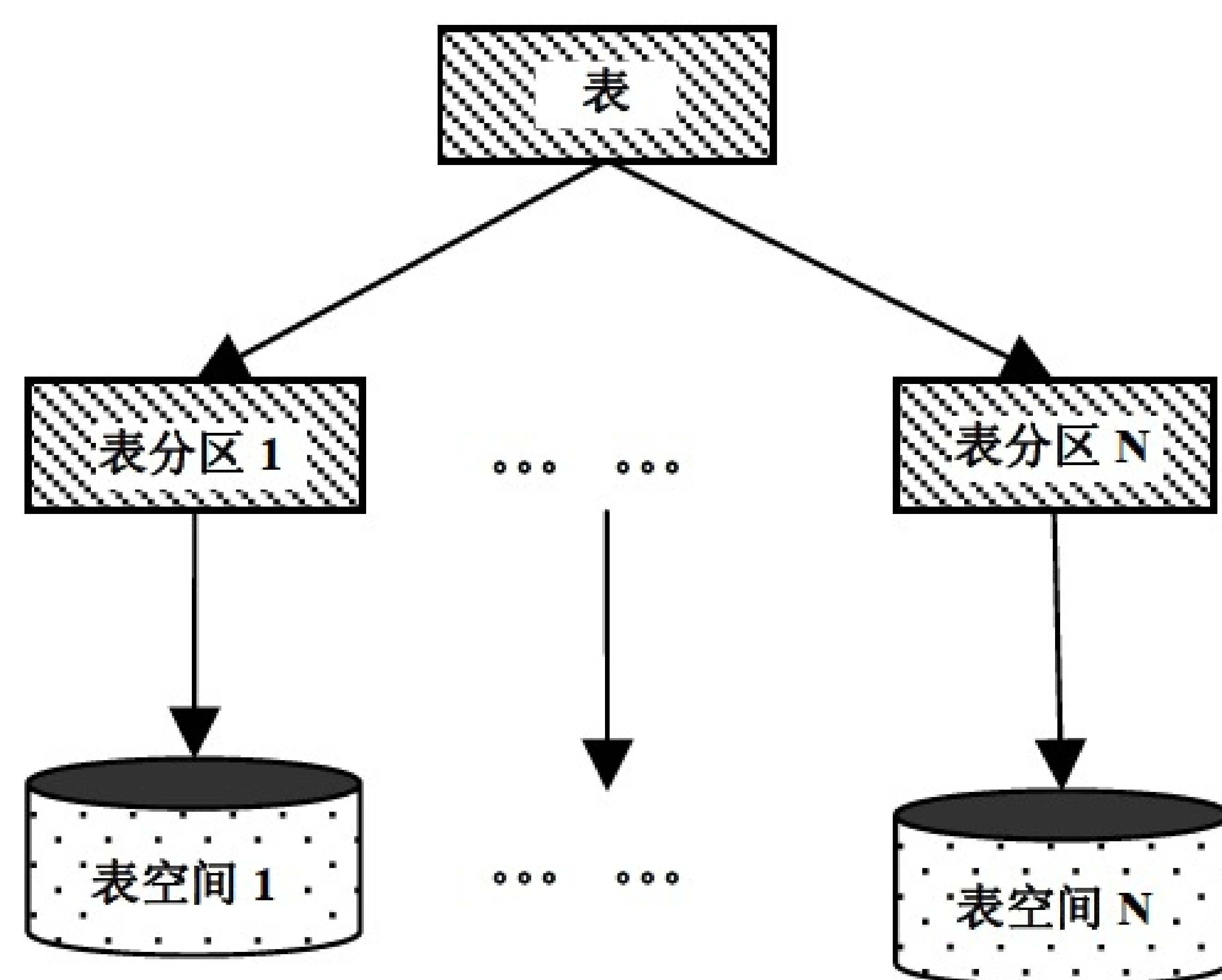


图 2-1　DB2 中的表分区特性

2.1.2　优点

表分区有如下许多优点：

1. 有效转入和转出

表分区功能提高了表数据的转入和转出效率，这可通过使用 ALTER TABLE 语句的 ATTACH PARTITION 和 DETACH PARTITION 子句来实现。通过转入分区表数据，可以方便地将新范围作为附加数据分区合并到分区表中。通过转出分区表数据，可以方便地从分区表中分离出某些范围的数据，以进行后续清除或归档处理。

2. 更容易管理大型表

由于可以在各个数据分区上执行管理任务，因此表级别管理更灵活。这些任务包括：拆离和重新连接数据分区、备份和复原各个数据分区以及重组各个索引。通过将花费较长时间的维护操作分解成一系列较小的操作，可以缩短这种维护操作的执行时间。例如，在将数据分区放置在单独的表空间中后，备份操作可以逐个处理数据分区。因此，可以一次备份分区表的一个数据分区。

3. 灵活的索引位置

DB2 可以将索引放置在不同的表空间中，从而允许对索引位置更精细地进行控制。这种新设计具有以下一些好处：

- 提高了删除索引和创建联机索引的性能。

- 能够针对每个表索引之间的任何表空间特征使用不同的值(例如，为了确保更好的空间利用率，对每个索引使用不同的页大小可能更合适)。
- 减少 IO 争用并提供对表索引数据更有效的并发访问。
- 删除各个索引时，空间将立即可供系统使用，而无须进行索引重组。
- 如果选择执行索引重组，可以重组单个索引。
- DMS 和 SMS 表空间都支持在不同于表的另一个位置使用索引。

4. 提高了商业智能样式查询的性能

增强了查询处理功能，能够根据查询谓词自动消除某些数据分区。此功能称为数据分区消除，可为许多决策支持查询带来好处。

2.1.3　分区表的基本用法

1. 创建基本表分区

创建基本分区表的语法非常简单，和创建普通表类似，只不过是在 CREATE TABLE 后面增加 PARTITION BY RANGE 的参数，定义分区表的范围。比如下面这个例子里将创建具有 4 个范围的 SALES 表，其中的范围包括边界值：

```
CREATE TABLE TEST.SALES
( ID                 INTEGER NOT NULL,
  SALES PERSON       VARCHAR(50),
  REGION             VARCHAR(50),
  SALES DATE         DATE)
  PARTITION BY RANGE(SALES DATE)
( PART PJAN STARTING '1/1/2012' ENDING '1/31/2012',
  PART PFEB STARTING '2/1/2012' ENDING '2/29/2012',
  PART PMAR STARTING '3/1/2012' ENDING '3/31/2012',
  PART PAPR STARTING '4/1/2012' ENDING '4/30/2012')
```

创建完表空间后，我们可以使用 describe 命令来查看分区表的结构，其中第一段显示了分区范围(包括边界)，第二段显示了分区的详细信息，包括分区名、表空间：

```
$db2 "describe DATA PARTITIONS for table test.sales show detail"

PartitionId Inclusive (y/n)                 Inclusive (y/n)
              Low Value                    High Value
----------- - ------------------------------ ----------------------------
          0 Y '2012-01-01'                  Y '2012-01-31'
          1 Y '2012-02-01'                  Y '2012-02-29'
```

```
          2 Y '2012-03-01'                   Y '2012-03-31'
          3 Y '2012-04-01'                   Y '2012-04-30'

  4 record(s) selected.

    PartitionId PartitionName                   TableSpId   PartObjId
IndexTblSpId LongTblSpId AccessMode             Status
    ----------- ------------------------------- ----------- ----------- ------
    0 PJAN                                    7          7           7           7 F
    1 PFEB                                    7          8           7           7 F
    2 PMAR                                    7          9           7           7 F
    3 PAPR                                    7         10           7           7 F

  4 record(s) selected.
```

AccessMode 表示分区的访问状态，主要有下面 4 种：

- D = 无数据移动
- F = 完全访问
- N = 无访问
- R = 只读访问

插入分区数据的语法和普通 insert 语法一样，但是对于未落入任何分区的值，insert 语句会返回报错信息。

```
    $db2 "insert into test.sales values (1,'James','US','2012-6-30-10.00.00')"
    DB21034E  The command was processed as an SQL statement because it was not
a valid Command Line Processor command.  During SQL processing it returned:
    SQL0327N  The row cannot be inserted into table "TEST.SALES" because it is
    outside the bounds of the defined data partition ranges.  SQLSTATE=22525

    $db2 "insert into test.sales values (1,'James','US','2012-4-30-10.00.00')"
    DB20000I  The SQL command completed successfully.
```

如果想知道当前数据落在哪个分区，那么可以使用 datapartitionnum 函数，用法如下：

```
    $db2 "select datapartitionnum(SALES DATE) as part id, SALES DATE from
test.sales"
    PART ID     SALES DATE
    ----------- ----------
              2 03/21/2012
              3 04/30/2012
  2 record(s) selected.
```

下面，为了更好地处理月份的不同天数，我们对这个例子做下改造，对结束值加

exclusive 参数：

```
CREATE TABLE TEST.SALES
( ID                INTEGER NOT NULL,
  SALES PERSON       VARCHAR(50),
  REGION            VARCHAR(50),
  SALES DATE         DATE)
  PARTITION BY RANGE(SALES DATE)
( STARTING '1/1/2012' ENDING '2/1/2012' exclusive,
  STARTING '2/1/2012' ENDING '3/1/2012' exclusive,
  STARTING '3/1/2012' ENDING '4/1/2012' exclusive,
  STARTING '4/1/2012' ENDING '5/1/2012' exclusive);
```

检查表定义，可见 Inclusive 列为 N：

```
$db2 "describe DATA PARTITIONS for table test.sales"

PartitionId Inclusive (y/n)                 Inclusive (y/n)
            Low Value                      High Value
----------- - ------------------------------- -------------------------
        0 Y '2012-01-01'                   N '2012-02-01'
        1 Y '2012-02-01'                   N '2012-03-01'
        2 Y '2012-03-01'                   N '2012-04-01'
        3 Y '2012-04-01'                   N '2012-05-01'

  4 record(s) selected.
```

为了更加简化创建脚本，我们可以采用 DB2 提供的具有生成范围的分区表来按月自动分区，语法如下所示：

```
CREATE TABLE TEST.SALES
( ID                INTEGER NOT NULL,
  SALES PERSON       VARCHAR(50),
  REGION            VARCHAR(50),
  SALES DATE         DATE)
  PARTITION BY RANGE(SALES DATE)
( STARTING MINVALUE,
  STARTING '1/1/2012' ENDING '12/31/2012'
             EVERY 1 MONTH,
 ENDING MAXVALUE);
```

检查表定义，可见 DB2 会自动按月建立分区。在这个例子里，我们增加了边界分区，所有 2012 年之前数据都属于分区 0，所有 2012 年之后数据都属于分区 13。

```
$db2 "describe DATA PARTITIONS for table test.sales"

PartitionId Inclusive (y/n)                    Inclusive (y/n)
            Low Value                          High Value
----------- - ------------------------------ - ------------------------------
          0 Y MINVALUE                       N '2012-01-01'
          1 Y '2012-01-01'                   N '2012-02-01'
          2 Y '2012-02-01'                   N '2012-03-01'
          3 Y '2012-03-01'                   N '2012-04-01'
          4 Y '2012-04-01'                   N '2012-05-01'
          5 Y '2012-05-01'                   N '2012-06-01'
          6 Y '2012-06-01'                   N '2012-07-01'
          7 Y '2012-07-01'                   N '2012-08-01'
          8 Y '2012-08-01'                   N '2012-09-01'
          9 Y '2012-09-01'                   N '2012-10-01'
         10 Y '2012-10-01'                   N '2012-11-01'
         11 Y '2012-11-01'                   N '2012-12-01'
         12 Y '2012-12-01'                   Y '2012-12-31'
         13 N '2012-12-31'                   Y MAXVALUE

  14 record(s) selected.
```

2. 定义分区表空间和分区键

下面，为了提高数据库性能，我们将不同的分区放到不同的表空间下。首先创建 6 个表空间，3 个数据表空间，3 个索引表空间：

```
    db2 "create tablespace ts_dat managed by database using (file
'/db2test/ts_dat' 100M)"
    db2 "create tablespace ts_dat1 managed by database using (file
'/db2test/ts_dat1' 100M)"
    db2 "create tablespace ts_dat2 managed by database using (file
'/db2test/ts_dat2' 100M)"
    db2 "create tablespace ts_idx managed by database using (file
'/db2test/ts_idx' 100M)"
    db2 "create tablespace ts_idx1 managed by database using (file
'/db2test/ts idx1' 100M)"
    db2 "create tablespace ts_idx2 managed by database using (file
'/db2test/ts_idx2' 100M)"
```

创建表 TEST.SALES，让每个分区分别使用这些表空间：

```
CREATE TABLE TEST.SALES
( ID              INTEGER NOT NULL,
```

```
  SALES_PERSON      VARCHAR(50),
  REGION           VARCHAR(50),
  SALES_DATE        DATE)
  IN ts_dat1,ts_dat2
  INDEX IN ts_idx
  PARTITION BY RANGE(SALES_DATE)
( STARTING MINVALUE,
  STARTING '1/1/2012' ENDING '12/31/2012'
              EVERY 1 MONTH,
 ENDING MAXVALUE);
```

执行 describe 命令，可见每个分区都顺序使用了这两个表空间：

```
$db2 "describe DATA PARTITIONS for table test.sales show detail"

PartitionId Inclusive (y/n)                    Inclusive (y/n)
            Low Value                        High Value
----------- - ------------------------------ ----------------------------
        0 Y MINVALUE                        N '2012-01-01'
        1 Y '2012-01-01'                    N '2012-02-01'
        2 Y '2012-02-01'                    N '2012-03-01'
        ... ...
       13 N '2012-12-31'                    Y MAXVALUE

  14 record(s) selected.


PartitionId PartitionName                    TableSpId  PartObjId
IndexTblSpId LongTblSpId AccessMode           Status
----------- ------------------------------ ---------- ---------- ------
0 PART0                                      9          4          9          9 F
1 PART1                                     10          4         10         10 F
2 PART2                                      9          5          9          9 F
3 PART3                                     10          5         10         10 F
4 PART4                                      9          6          9          9 F
 ... ...

  14 record(s) selected.
```

类似的，我们也可以在分区后面加表空间名称，显式地为分区指定所在的表空间。对于建立分区语句里没有指定表空间的分区，使用 CREATE TABLE 里指定的表空间。

```
CREATE TABLE TEST.SALES
( ID                INTEGER NOT NULL,
```

```
      SALES PERSON      VARCHAR(50),
      REGION           VARCHAR(50),
      SALES DATE        DATE)
      IN TS_DAT
      INDEX IN TS_IDX
      PARTITION BY RANGE(SALES DATE)
    ( PART PJAN STARTING '1/1/2012' ENDING '1/31/2012' IN TS_DAT1 INDEX IN
TS_IDX1,
      PART PFEB STARTING '2/1/2012' ENDING '2/29/2012' IN TS_DAT1 INDEX IN
TS_IDX1,
      PART PMAR STARTING '3/1/2012' ENDING '3/31/2012' IN TS_DAT2 INDEX IN
TS IDX2,
      PART PAPR STARTING '4/1/2012' ENDING '4/30/2012' );

    $db2 "describe DATA PARTITIONS for table test.sales show detail"

    PartitionId Inclusive (y/n)                  Inclusive (y/n)
                Low Value                       High Value
    ----------- - ------------------------------ ---------------------------
            0 Y '2012-01-01'                     Y '2012-01-31'
            1 Y '2012-02-01'                     Y '2012-02-29'
            2 Y '2012-03-01'                     Y '2012-03-31'
            3 Y '2012-04-01'                     Y '2012-04-30'

      4 record(s) selected.

    PartitionId PartitionName                    TableSpId   PartObjId
IndexTblSpId LongTblSpId AccessMode              Status
    ----------- ------------------------------- ----------- ----------- ------
    0 PJAN                                   9          4          12          9 F
    1 PFEB                                   9          5          12          9 F
    2 PMAR                                  10          4          13         10 F
    3 PAPR                                   8          4           8          8 F

      4 record(s) selected.
```

表分区键是一个或多个表列的有序集合。表分区键列中的值用来确定每个表行所属的数据分区。

选择有效的表分区键列对于充分利用表分区功能的优点来说十分关键。下列准则可以帮助为分区表选择最有效的表分区键列。

- 将范围详细程度定义为与数据转出相匹配。最常见的情况是使用星期、月份或季度。
- 将范围定义成与数据转入大小相匹配。最常见的情况是根据日期或时间列对数据进行分区。
- 根据有益于消除分区的列进行分区。

表分区键支持的数据类型如表 2-1 所示。

表 2-1　表分区键支持的数据类型

数据类型列 1	数据类型列 2
SMALLINT	INTEGER
INT	BIGINT
FLOAT	REAL
DOUBLE	DECIMAL
DEC	DECFLOAT
NUMERIC	NUM
CHARACTER	CHAR
VARCHAR	DATE
TIME	GRAPHIC
VARGRAPHIC	CHARACTER VARYING
TIMESTAMP	CHAR VARYING
CHARACTER FOR BIT DATA	CHAR FOR BIT DATA
VARCHAR FOR BIT DATA	CHARACTER VARYING FOR BIT DATA
CHAR VARYING FOR BIT DATA	用户定义的类型(单值)

表分区键不支持的数据类型如下：

- 用户定义的类型(结构化)
- LONG VARCHAR
- LONG VARCHAR FOR BIT DATA
- BLOB
- BINARY LARGE OBJECT

- CLOB
- CHARACTER LARGE OBJECT
- DBCLOB
- LONG VARGRAPHIC
- REF
- C 变长字符串
- Pascal 变长字符串
- XML

如果选择使用 CREATE TABLE 语句的 EVERY 子句来自动生成数据分区，那么只能将一列用作表分区键。如果选择通过在 CREATE TABLE 语句的 PARTITION BY 子句中指定每个范围来手动生成数据分区，那么可以将多个列用作表分区键，如以下示例所示：

```
CREATE TABLE TEST.SALES
( ID                INTEGER NOT NULL,
  SALES PERSON      VARCHAR(50),
  REGION            VARCHAR(50),
  SALES YEAR        INT,
  SALES MONTH       INT)
  PARTITION BY RANGE(SALES YEAR, SALES MONTH)
( STARTING (2012,1) ENDING (2012,6),
  ENDING (2012, 12),
  ENDING (2013, 6),
  ENDING (2013, 12));

$db2 "describe DATA PARTITIONS for table test.sales"

PartitionId Inclusive (y/n)                 Inclusive (y/n)
              Low Value                    High Value
----------- - ------------------------------- ----------------------------
          0 Y 2012,1                       Y 2012,6
          1 N 2012,6                       Y 2012,12
          2 N 2012,12                      Y 2013,6
          3 N 2013,6                       Y 2013,12

  4 record(s) selected.
```

这将生成 4 个数据分区，即 2012 年和 2013 年的每半年就有一个数据分区。

分区键还可以指定自动生成的列，比如在下面的例子里，SALES_MONTH 是由 SALES_DATE 通过 month 函数生成的，而后按照月份生成分区。

```
CREATE TABLE TEST.SALES
( ID            INTEGER NOT NULL,
  SALES PERSON    VARCHAR(50),
  REGION         VARCHAR(50),
  SALES DATE      DATE,
  SALES MONTH     GENERATED ALWAYS AS (month(SALES DATE)))
  PARTITION BY RANGE(SALES MONTH)
( STARTING FROM 1 ENDING AT 12 EVERY 1 );

$db2 "describe DATA PARTITIONS for table test.sales"

PartitionId Inclusive (y/n)                 Inclusive (y/n)
            Low Value                      High Value
----------- - ------------------------------ ---------------------------
          0 Y 1                          N 2
          1 Y 2                          N 3
          2 Y 3                          N 4
          3 Y 4                          N 5
          4 Y 5                          N 6
          5 Y 6                          N 7
          6 Y 7                          N 8
          7 Y 8                          N 9
          8 Y 9                          N 10
          9 Y 10                         N 11
         10 Y 11                         N 12
         11 Y 12                         Y 12

  12 record(s) selected.
```

3. 执行计划

对于查询，如果查询里命中了分区键，那么 DB2 优化器会先计算条件属于哪个分区，然后只对该分区进行扫描。下面我们来通过分析表分区的执行计划来解释一下分区表是如何提高查询性能的。

首先创建如下没有查询条件的 SQL：

```
select count(*) from test.sales group by region
```

使用 db2expln 分析执行计划，所有分区都会被扫描：

```
$db2expln -d sample -t -f sel1.sql
… …
Access Table Name = TEST.SALES  ID = -6,-32768
```

```
    |  #Columns = 1
    |  Data-Partitioned Table
    |  Skip Inserted Rows
    |  Avoid Locking Committed Data
    |  Currently Committed for Cursor Stability
    |  May participate in Scan Sharing structures
    |  Scan may start anywhere and wrap, for completion
    |  Fast scan, for purposes of scan sharing management
    |  Scan can be throttled in scan sharing management
    |  All data partitions will be accessed
    |  Relation Scan
    |  |  Prefetch: Eligible
    |  Lock Intents
    |  |  Table: Intent Share
    |  |  Row  : Next Key Share
    |  Sargable Predicate(s)
    |  |  Insert Into Sorted Temp Table  ID = t1
    |  |  |  #Columns = 2
    |  |  |  #Sort Key Columns = 1
    |  |  |  |  Key 1: REGION (Ascending)
    |  |  |  Sortheap Allocation Parameters:
    |  |  |  |  #Rows     = 1.000000
    |  |  |  |  Row Width = 16
    |  |  |  Piped
    |  |  |  Buffered Partial Aggregation
    Sorted Temp Table Completion  ID = t1
    Access Temp Table  ID = t1
    |  #Columns = 2
    |  Relation Scan
    |  |  Prefetch: Eligible
    Final Aggregation
    |  Group By
    |  Column Function(s)
    Return Data to Application
    |  #Columns = 1
```

修改上面的 SQL，增加 sales_date 的查询条件：

```
    select count(*) from test.sales where sales date='2012-02-11-11.11.11'
group by region
```

使用 db2expln 生成执行计划，可见 DB2 优化器首先按照分区条件进行过滤，只在 1 号表分区上进行扫描，因此这样做会必然减少扫描的数据量，提高 SQL 性能。

```
    Access Table Name = TEST.SALES  ID = -6,-32768
```

```
|  #Columns = 1
|  Data-Partitioned Table
|  Skip Inserted Rows
|  Avoid Locking Committed Data
|  Currently Committed for Cursor Stability
|  May participate in Scan Sharing structures
|  Scan may start anywhere and wrap, for completion
|  Fast scan, for purposes of scan sharing management
|  Scan can be throttled in scan sharing management
|  Data Partition Elimination Info:
|  |  Range 1:
|  |  |  #Key Columns = 1
|  |  |  |  Start Key: Inclusive Value
|  |  |  |  |  1: 2012-02-11
|  |  |  |  Stop Key: Inclusive Value
|  |  |  |  |  1: 2012-02-11
|  Active Data Partitions: 1
|  Relation Scan
|  |  Prefetch: Eligible
|  Lock Intents
|  |  Table: Intent Share
|  |  Row  : Next Key Share
|  Sargable Predicate(s)
|  |  #Predicates = 1
|  |  Insert Into Sorted Temp Table  ID = t1
|  |  |  #Columns = 1
|  |  |  #Sort Key Columns = 1
|  |  |  |  Key 1: REGION (Ascending)
|  |  |  Sortheap Allocation Parameters:
|  |  |  |  #Rows     = 1.000000
|  |  |  |  Row Width = 12
|  |  |  Piped
Sorted Temp Table Completion  ID = t1
Access Temp Table  ID = t1
|  #Columns = 1
|  Relation Scan
|  |  Prefetch: Eligible
|  Sargable Predicate(s)
|  |  Predicate Aggregation
|  |  |  Group By
|  |  |  Column Function(s)
Aggregation Completion
|  Group By
```

```
|  Column Function(s)
Return Data to Application
|  #Columns = 1
```

4. 将现有表和视图迁移到分区表

在 DB2 中，可以通过 3 种方法将现有的表或视图迁移到分区表：

- 迁移正规表时，创建新的空分区表并使用 LOAD from CURSOR 将数据从旧表直接移到分区表中，而不执行任何中间步骤。
- 迁移常规表时，使用 EXPORT 实用程序或高性能卸载来卸载源表，创建新的空分区表并使用 LOAD 命令填充空的分区表。
- 迁移 UNION ALL 视图时，创建带有单一虚拟数据分区的分区表，然后连接所有的表。

第一种方法是比较方便和常用的，因此这里通过具体示例来描述一下第一种方法。

(1) 我们先定义非分区表 TBL_LMSGLOG 和分区表 TBL_LMSGLOG_TEST ，DDL 如下：

```
CREATE TABLE "IBPS    "."TBL_LMSGLOG"  (
               "BUSIID" VARCHAR(60) NOT NULL ,
               "SENDER" VARCHAR(10) NOT NULL ,
               "SENDID" VARCHAR(128) NOT NULL ,
               "OPERCD" VARCHAR(20) ,
               "SENDTM" TIMESTAMP ,
               "REQUINFO" VARCHAR(4000) ,
               "DEALCD" VARCHAR(10) ,
               "DEALINF" VARCHAR(500) ,
               "UPDATETIME" TIMESTAMP WITH DEFAULT CURRENT TIMESTAMP ,
               "SNDRCVIND" VARCHAR(1) )
              IN "TBS_UPP_8K" ;

ALTER TABLE "IBPS    "."TBL_LMSGLOG"
       ADD CONSTRAINT "TBL LMSGLOG PRI" PRIMARY KEY
              ("BUSIID",
               "SENDER",
               "SENDID");
```

(2) 选择 UPDATETIME 字段为分区键，创建分区表 TBL_LMSGLOG_TEST：

```
CREATE TABLE "IBPS    "."TBL_LMSGLOG_TEST"  (
               "BUSIID" VARCHAR(60) NOT NULL ,
               "SENDER" VARCHAR(10) NOT NULL ,
```

```
                   "SENDID" VARCHAR(128) NOT NULL ,
                   "OPERCD" VARCHAR(20) ,
                   "SENDTM" TIMESTAMP ,
                   "REQUINFO" VARCHAR(4000) ,
                   "DEALCD" VARCHAR(10) ,
                   "DEALINF" VARCHAR(500) ,
                   "UPDATETIME" TIMESTAMP WITH DEFAULT CURRENT TIMESTAMP ,
                   "SNDRCVIND" VARCHAR(1) )
                  IN "TBS_UPP_8K" PARTITION BY RANGE("UPDATETIME")
                  (PART "PART0" STARTING(MINVALUE) ,
                  PART "PART201101" STARTING('2011-01-01 00:00:00')
ENDING('2011-02-01 00:00:00') EXCLUSIVE ,
                  PART "PART201102" STARTING('2011-02-01 00:00:00')
ENDING('2011-03-01 00:00:00') EXCLUSIVE ,
                  PART "PART201103" STARTING('2011-03-01 00:00:00')
ENDING('2011-04-01 00:00:00') EXCLUSIVE ,
                  PART "PART201104" STARTING('2011-04-01 00:00:00')
ENDING('2011-05-01 00:00:00') EXCLUSIVE ,
                  PART "PART201105" STARTING('2011-05-01 00:00:00')
ENDING('2011-06-01 00:00:00') EXCLUSIVE ,
                  PART "PART201106" STARTING('2011-06-01 00:00:00')
ENDING('2011-07-01 00:00:00') EXCLUSIVE ,
    ... ...
                  PART "PART37" ENDING(MAXVALUE) );
```

(3) 利用 load cursor 将数据从原表导入到分区表中：

```
    select current time from syscat.tables fetch first 1 rows only;
    DECLARE c1 CURSOR FOR SELECT * FROM IBPS.TBL_LMSGLOG;
    LOAD FROM c1 of CURSOR INSERT INTO IBPS.TBL_LMSGLOG TEST NONRECOVERABLE;
    select current time from syscat.tables fetch first 1 rows only;
```

(4) 删除原表主键，进行原表和分区表重命名，创建分区表主键：

```
    --删除原表主键
    db2 "alter table TBL LMSGLOG drop primary key"
    --进行原表和分区表重命名
    db2 "rename table TBL LMSGLOG to  TBL_LMSGLOG OLD"
    db2 "rename table TBL LMSGLOG TEST to  TBL_LMSGLOG"
    --创建 primary key
    ALTER TABLE "IBPS    "."TBL LMSGLOG"
          ADD CONSTRAINT "TBL_LMSGLOG PRI" PRIMARY KEY
                 ("BUSIID",
                  "SENDER",
```

```
                    "SENDID");
```

(5) 验证对象有效性，针对失效的 package 进行 rebind：

```
    --验证表或视图的状态
    db2 "select substr(tabschema,1,10) as tabschema,substr(tabname,1,30) as
tabname,status from syscat.tables where status != 'N' with ur"
    --验证 package 的状态
    db2 "select substr(PKGSCHEMA,1,10) as PKGSCHEMA,substr(PKGNAME,1,30) as
PKGNAME,VALID from syscat.PACKAGES  where VALID !='Y' with ur"
    --验证 routine 的状态
    db2 "select substr(ROUTINESCHEMA,1,10) as
ROUTINESCHEMA,substr(ROUTINENAME,1,30) as ROUTINENAME,ROUTINETYPE,VALID from
syscat.ROUTINES  where VALID!='Y' and VALID !='' with ur"
```

(6) 针对分区表做 runstats：

```
    db2 "runstats on table ibps.TBL LMSGLOG with distribution and detailed
indexes all"
```

2.1.4 分区表的管理

1. 分区转出

除了可以支持更大的表，增加性能外，分区表的另外一个好处是可以灵活快速地将分区转入转出。分区的转出操作可以将某个范围分区分成独立的表，提交后在主表内的这部分数据立刻变为不可视。此过程并不会造成数据的移动，因此速度很快，对数据库干扰很少，通过转出分区表数据，可以方便地从分区表中分离出某些范围的数据。一旦将数据分区拆离成单独的表，就可以通过多种方法处理这些表。可以删除单独的表(这样就破坏了数据分区中的数据)；可以对它们进行归档或者以别的方式将它们作为单独的表使用；将它们连接到另一个分区表(例如历史表)；也可以对它们进行操纵、清理和变换以及将它们重新连接到原始分区表或另一分区表。

利用 DB2 V9.7 SP1 及更高发行版，在使用带 DETACH PARTITION 子句的 ALTER TABLE 语句从分区表拆离数据分区时，源分区表将保持联机，并且对该表运行的查询可继续运行。通过下列两个阶段可将要拆离的数据分区转换为独立的表：

(1) ALTER TABLE...DETACH PARTITION 操作在逻辑上将数据分区从分区表拆离。

(2) 异步分区拆离任务将在逻辑上已拆离的分区转换为独立的表。

如果有任何需要根据已拆离数据分区进行递增维护的从属表(这些从属表称为已拆离的从属表)，那么仅在对所有已拆离的从属表运行 SET INTEGRITY 语句之后，异步分区拆离任务才会开始。

如果没有已拆离的从属表，那么在发出 ALTER TABLE...DETACH PARTITION 语句的事务落实之后，异步分区拆离任务就会开始。

分区的转出使用语法 ALTER TABLE … DETACH。例如，创建如下包含 4 个分区的销售表，按月进行分区：

```
    CREATE TABLE TEST.SALES
    ( ID                INTEGER NOT NULL,
      SALES PERSON       VARCHAR(50),
      REGION            VARCHAR(50),
      SALES DATE         DATE)
      PARTITION BY RANGE(SALES DATE)
    ( STARTING '1/1/2012' ENDING '2/1/2012' exclusive,
      STARTING '2/1/2012' ENDING '3/1/2012' exclusive,
      STARTING '3/1/2012' ENDING '4/1/2012' exclusive,
      STARTING '4/1/2012' ENDING '5/1/2012' exclusive);
```

插入 4 条数据，每个分区一条数据：

```
    $db2 "insert into test.sales values (1,'James','US','2012-1-21-08.11.00')"
    $db2 "insert into test.sales values (2,'Peter','US','2012-2-11-19.20.00')"
    $db2 "insert into test.sales values (3,'Kitty','US','2012-3-20-08.08.11')"
    $db2 "insert into test.sales values (4,'Kary','US','2012-4-30-22.32.34')"
    $db2 "select datapartitionnum(SALES DATE) as part id, id,
substr(sales person,1,10) as sales person, substr(region,1,10) as
region,SALES DATE from test.sales"
    PART ID         ID          SALES PERSON  REGION     SALES DATE
    ----------- ----------- ------------ ---------- ----------
              0     1          James         US         01/21/2012
              1     2          Peter         US         02/11/2012
              2     3          Kitty         US         03/20/2012
              3     4          Kary          US         04/30/2012

      4 record(s) selected.
```

从 SALES 表中转出 1 月份的数据到 SALES_JAN 表：

```
    $db2 "alter table test.sales detach partition part0 into table test.sales jan"
    DB20000I  The SQL command completed successfully.

    db2 "select datapartitionnum(SALES DATE) as part id, id,
substr(sales person,1,10) as sales person, substr(region,1,10) as
region,SALES DATE from test.sales"
```

```
    PART ID      ID          SALES PERSON REGION     SALES DATE
    ----------- ----------- ------------ ---------- ----------
              0 2           Peter        US         02/11/2012
              1 3           Kitty        US         03/20/2012
              2 4           Kary         US         04/30/2012

      3 record(s) selected.

    db2 "select datapartitionnum(SALES DATE) as part id, id,
substr(sales person,1,10) as sales person, substr(region,1,10) as
region,SALES DATE from test.sales jan"

    PART ID      ID          SALES PERSON  REGION      SALES DATE
    ----------- ----------- ------------  ----------  ----------
              0 1           James         US          01/21/2012

      1 record(s) selected.
```

此时，detach 会迅速将 1 月份的分区 PART0 转为表 SALES_JAN，并且主表 SALES 无法再查询到 PART0 的数据。

2. 分区转入

分区的转入操作可以将现存表作为新的范围分区加到主表里，提交后该分区的数据在主表里立刻变为可视，此过程同样不会触发数据移动，因此速度非常快，对数据库影响非常小。我们可以紧接前面的例子将 SALES_JAN 表转入 SALES 表内：

```
    $db2 "alter table test.sales attach partition part jan STARTING '1/1/2012'
ENDING '2/1/2012' exclusive FROM TABLE test.sales jan"
    SQL3601W  The statement caused one or more tables to automatically be placed
    in the Set Integrity Pending state.  SQLSTATE=01586
```

注意此时 sales 表里该分区不可用，处于 set integrity pending 状态，我们可以通过查看 syscat.tables 里的 STATUS 字段得到表的状态：

```
    $db2 "select status from syscat.tables where tabname='SALES' and
tabschema='TEST'"

    STATUS
    ------
    C

      1 record(s) selected.
```

```
$db2 "select * from test.sales"

ID    SALES PERSON     REGION    SALES DATE
---- --------------- ---------- ----------
 2    Peter            US       02/11/2012
 3    Kitty            US       03/20/2012
 4    Kary             US       4/30/2012

  3 record(s) selected.
```

检查该分区，状态为 N，代表不可用：

```
$db2 describe data partitions for table test.sales show detail

PartitionId Inclusive (y/n)                  Inclusive (y/n)
            Low Value                       High Value
----------- - ------------------------------- ---------------------------
        0 Y '2012-01-01'                   N '2012-02-01'
        1 Y '2012-02-01'                   N '2012-03-01'
        2 Y '2012-03-01'                   N '2012-04-01'
        3 Y '2012-04-01'                   N '2012-05-01'

  4 record(s) selected.

PartitionId PartitionName                TableSpId  PartObjId  IndexTblSpId
LongTblSpId AccessMode                   Status
----------- ------------------------------ ----------- ----------- ------------ - ------
0 PART JAN                              7          7          7          7 N A
1 PART1                                 7          8          7          7 F
2 PART2                                 7          9          7          7 F
3 PART3                                 7         10          7          7 F

  4 record(s) selected.
```

这时需要我们手工对 TEST.SALES 表执行完整性检查，此过程将在新连接的分区上完成范围检查，并强制执行存在的任何约束。完成后，新连接的数据便在数据库中可视：

```
$db2 "set integrity for test.sales immediate checked"
DB20000I  The SQL command completed successfully.
$db2 "select * from test.sales"

ID  SALES PERSON          REGION     SALES DATE
 ----- --------------------- ----------- -----------
```

```
    1   James                   US        01/21/2012
    2   Peter                   US        02/11/2012
    3   Kitty                   US          03/20/2012
    4   Kary                    US           04/30/2012

      4 record(s) selected.
```

3. 分区索引

DB2 中实现了分区索引(partitioned index)的特性。与数据分区类似，表上的索引数据也将由多个索引分区(index partition)来存放。每个索引分区存放对应数据分区的索引数据。传统的未分区索引被称为非分区索引(nonpartitioned index)，也称为全局索引(global index)，相应的分区索引也可被称为本地索引(local index)，如图 2-2 所示。

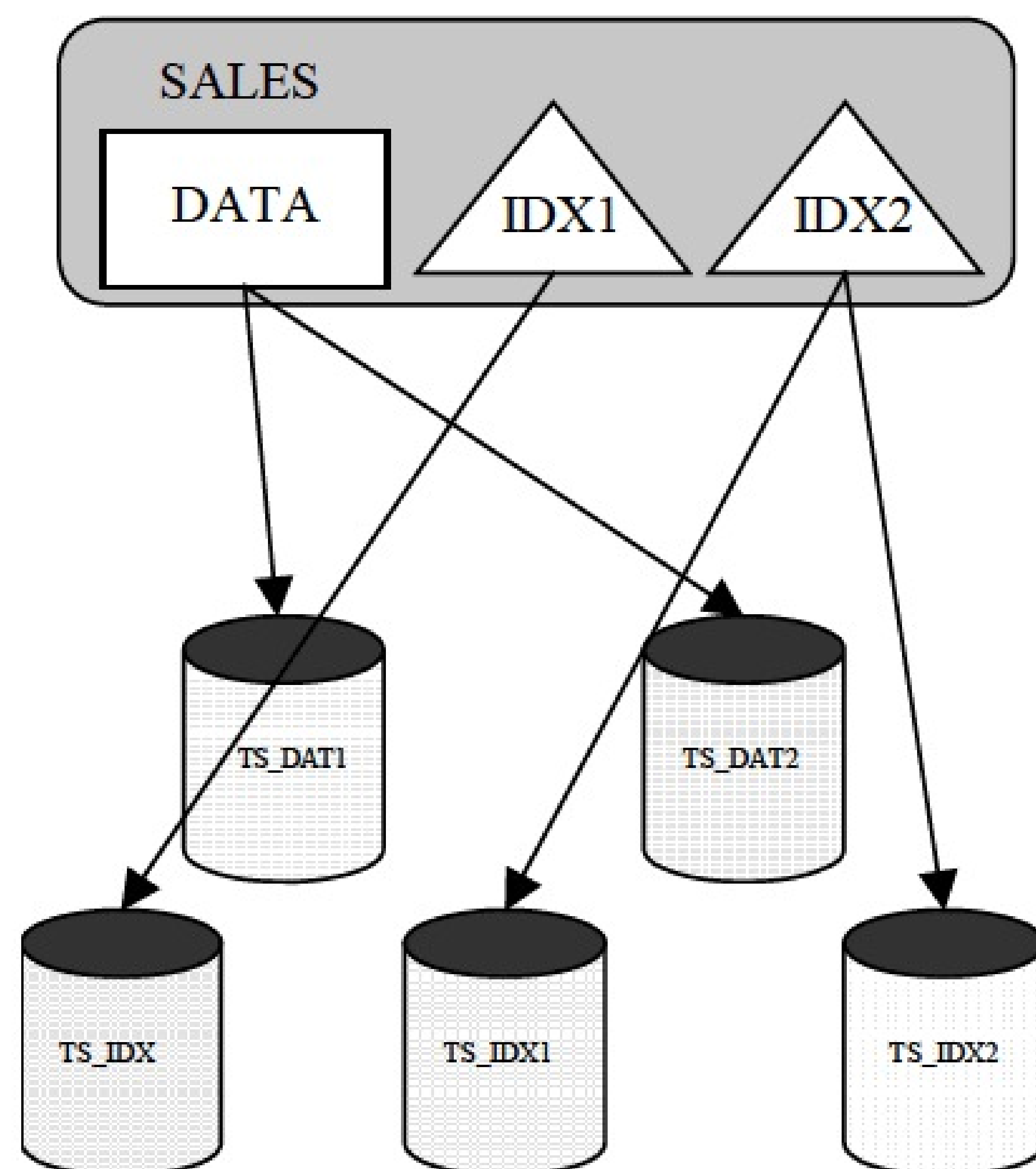


图 2-2　全局索引和分区索引

分区索引带来的一个显著优势在于，在使用 ALTER TABLE ATTACH PARTITION 和 DETAICH PARTITION 命令进行数据转入(roll-in)和转出(roll-out)时，使用分区索引能够很大程度地提高性能。

在 DB2 V9.7 FP1 以及之后的版本中，以下类型的索引不能是分区索引，只能是非分区索引：

- 空间数据(spatial data)索引
- XML path index(系统生成索引)

对于分区表 TEST.SALES，有两个索引 IDX1 和 IDX2，其中 IDX1 只存储在 TS_IDX

表空间中，因此是全局索引；IDX2 则被分别存储在 TS_IDX1 和 TS_IDX2 中，因此属于分区索引。我们可以使用下面的 SQL 语句来创建该分区索引：

```
    CREATE TABLE TEST.SALES
    ( ID                  INTEGER NOT NULL,
      SALES PERSON        VARCHAR(50),
      REGION              VARCHAR(50),
      SALES DATE          DATE)
      IN TS DAT
      INDEX IN TS IDX
      PARTITION BY RANGE(SALES DATE)
    ( PART PJAN STARTING '1/1/2012' ENDING '1/31/2012' IN TS DAT1 INDEX IN
TS_IDX1,
      PART PFEB STARTING '2/1/2012' ENDING '2/29/2012' IN TS DAT1 INDEX IN
TS_IDX1,
      PART PMAR STARTING '3/1/2012' ENDING '3/31/2012' IN TS_DAT2 INDEX IN
TS IDX2,
      PART PAPR STARTING '4/1/2012' ENDING '4/30/2012' IN TS_DAT2 INDEX IN
TS IDX2);

    CREATE INDEX TEST.IDX1 on TEST.SALES(ID) NOT PARTITIONED;
    CREATE INDEX TEST.IDX2 on TEST.SALES(SALES_DATE) PARTITIONED;
```

DB2 使用 NOT PARTITIONED 和 PARTITIONED 关键字来指定索引是否分区，如果不加的话，默认为分区索引。

```
    $db2 describe INDEXES FOR TABLE test.sales

    Index  schema       Index name       Unique rule      Number of columns
    Index  type        Index partitioning
    ------------------------------- ------------------- ---------------------
    TEST                         IDX1             D                       1
RELATIONAL DATA           N
    TEST                         IDX2             D                       1
RELATIONAL DATA           P

      2 record(s) selected.
```

下面我们再来看一下分区索引的执行计划，首先创建如下 SQL：

```
    select count(*) from test.sales where sales date between
'2012-02-11-11.11.11' and '2012-03-11-11.11.11'
```

接着用 db2expln 工具生成如下执行计划：

```
Access Table Name = TEST.SALES  ID = -6,-32768
|  Index Scan:  Name = TEST.IDX2  ID = 2
|  |  Regular Index (Not Clustered)
|  |  Index Columns:
|  |  |  1: SALES_DATE (Ascending)
|  #Columns = 0
|  Data-Partitioned Table
|  Skip Inserted Rows
|  Avoid Locking Committed Data
|  Currently Committed for Cursor Stability
|  Forward scan through data partitions (local index)
|  Data Partition Elimination Info:
|  |  Range 1:
|  |  |  #Key Columns = 1
|  |  |  |  Start Key: Inclusive Value
|  |  |  |  |  1: 2012-02-11
|  |  |  |  Stop Key: Inclusive Value
|  |  |  |  |  1: 2012-03-11
|  Active Data Partitions: 1-2
|  #Key Columns = 1
|  |  Start Key: Inclusive Value
|  |  |  |  1: 2012-02-11
|  |  Stop Key: Inclusive Value
|  |  |  |  1: 2012-03-11
|  Index-Only Access
|  Index Prefetch: None
|  Lock Intents
|  |  Table: Intent Share
|  |  Row   : Next Key Share
|  Sargable Index Predicate(s)
|  |  Predicate Aggregation
|  |  |  Column Function(s)
Aggregation Completion
|  Column Function(s)
Return Data to Application
|  #Columns = 1
```

注意：

这条 SQL 语句执行了 SALES 表的分区索引扫描。在说明输出中要注意的是关于表分区，访问的表是被分区的，将执行分区排除功能以及删除活动数据分区的值。在本例中，

活动的数据分区为 1-2。这里引用的是 syscat.datapartitions 中的序列号(seqno)而不是 describe data partitions 命令中的 PartitionId。

2.1.5　分区重组

DB2 V9.7 以及之前版本的重组命令只能支持整个表的重组，而不能支持单个分区的重组。这导致只要有一个分区上的数据需要重组，就不得不对整个表上的所有数据都进行锁定和重组，这样重组操作的性能就会降低，同时也影响了表上其他事务的运行。在 DB2 V9.7 FP1 里，重组命令引入了对单个分区进行重组的新特性。这个新特性使得用户可以仅对某一个分区进行重组，并仅在该分区上对其他事务的读写权限进行限制。这样可以最大程度缩小重组命令对其他事务的影响，提高事务的并发度。这样当用户使用分区重组时，在表上没有非分区索引的情况下，重组命令将完全只在一个分区上进行，从而对其他分区上的事务没有任何影响。比如下面这个例子：

```
$db2 "reorg table test.sales allow read access ON DATA PARTITION part jan"
DB20000I  The REORG command completed successfully.

$db2 "reorg table test.sales allow no access ON DATA PARTITION part jan"
DB20000I  The REORG command completed successfully.

$db2 "reorg table test.sales inplace ON DATA PARTITION part jan"
SQL2216N  SQL error "-1548" occurred while reorganizing a database table or
its indexes.

$db2 "create index test.idx1 on test.sales(id) partitioned"
DB20000I  The SQL command completed successfully.

$db2 "reorg indexes all for table test.sales ON DATA PARTITION part jan"
DB20000I  The REORG command completed successfully.
```

分区重组命令里增加了 DATA PARTITION 子句和读写控制选项，如下所示：

```
REORG TABLE tabname [ALLOW NO/READ ACCESS] [ON DATA PARTITION partname]

REORG INDEXES ALL FOR TABLE tabname [ALLOW NO/READ/WRITE ACCESS] [ON DATA
PARTITION partname]
```

表 2-2 和表 2-3 列出了 DB2 分区重组命令里各选项搭配的实现效果。

表 2-2　DB2 V9.7 分区重组命令 REORG TABLE 里各选项搭配的实现效果

指 定 分 区	默认读写控制选项	ALLOW NO ACCESS	ALLOW READ ACCESS
ON DATA PARTITION part_i (没有非分区索引)	默认为 ALLOW READ ACCESS	part_i：不可访问 part_other：可读可写	part_i：仅可读 part_other：可读可写
ON DATA PARTITION part_i (有非分区索引存在)	默认为 ALLOW NO ACCESS	所有分区均不可访问	返回 SQL1548N 错误
未指定单个分区 (全表重组)	默认为 ALLOW NO ACCESS	所有分区均不可访问	返回 SQL1548N 错误

表 2-3　索引分区重组命令 REORG INDEXES ALL 里各选项搭配的实现效果

指 定 分 区	默认读写控制选项	ALLOW NO ACCESS	ALLOW READ ACCESS ALLOW WRITE ACCESS
ON DATA PARTITION part_i	默认为 ALLOW READ ACCESS	part_i：不可访问 part_other：可读可写	part_i：仅可读或可写 part_other：可读可写
未指定单个分区 (全表索引重组)	默认为 ALLOW NO ACCESS	所有分区均不可访问	返回 SQL1548N 错误

2.1.6　分区表 detach 的常见问题

1. 分区主表关联有强制外键约束

问题描述：从属表里创建了强制外键约束并关联到主表，此时是不能对主表进行 detach 操作的。比如下面的 SQL 语句：

```
    CREATE TABLE TESTPT P ( C1 INTEGER  NOT NULL , C2 TIMESTAMP  NOT NULL , C3
VARCHAR (10)  NOT NULL   ) PARTITION BY RANGE (C2 NULLS LAST)  (PARTITION PART1
STARTING FROM ('2011-06-18 00:00:00') INCLUSIVE ENDING AT ('2011-06-18 23:59:59')
EXCLUSIVE, PARTITION PART2 STARTING FROM ('2011-06-19 00:00:00') INCLUSIVE
ENDING AT ('2011-06-19 23:59:59') EXCLUSIVE , PARTITION PART3 STARTING FROM
('2011-06-20 00:00:00') INCLUSIVE ENDING AT ('2011-06-20 23:59:59') EXCLUSIVE,
PARTITION PART4 STARTING FROM ('2011-06-21 00:00:00')INCLUSIVE ENDING AT
(MAXVALUE) EXCLUSIVE) ;

    ALTER TABLE TESTPT P ADD CONSTRAINT CC1308539592563 PRIMARY KEY ( C1) ;
```

```
    CREATE TABLE TESTPT C ( C1 INTEGER  NOT NULL , C2 INTEGER  NOT NULL , C3 VARCHAR
(10)  NOT NULL  , CONSTRAINT CC1308539667273 FOREIGN KEY (C2) REFERENCES TESTPT P
(C1)  ON DELETE NO ACTION ON UPDATE NO ACTION ENFORCED  ENABLE QUERY
OPTIMIZATION  ) ;
```

插入一定的数据后，对主表进行 detach 操作会报如下错误：

```
    db2 "alter table testpt p detach partition part1 into table testpt p 1"
    DB21034E  The command was processed as an SQL statement because it was not
a valid Command Line Processor command.  During SQL processing it returned:
SQL0270N  Function not supported (Reason code = "92").  SQLSTATE=42997
```

解决方法：对于有主外键约束的情况，做 detach 的时候一定要慎重！因为虽然可以将外键约束从 enforced 改为 not enforced 或者删除外键约束，然后重新执行 detach，但是 detach 成功后，有可能出现从属表和主表数据不一致的问题，导致无法将外键约束改为 enforced，这时需要执行 delete 命令，删除从属表里违反外键约束的行，然后重新执行 enforced 命令。

```
    db2 "alter table testpt c alter foreign key CC1308539667273 not enforced"
    db2 "alter table testpt p detach partition part1 into table testpt p 1"
    db2 commit
    db2 "delete from testpt c where c1 in (select c1 from testpt p 1)"
    db2 "alter table testpt_c alter foreign key CC1308539667273 not forced"
```

2. 分区主表含有非自动刷新的 MQT

问题描述：如果分区主表包含非自动刷新的 MQT，那么对主表做 detach 后，MQT 并不会自动刷新，需要手工执行 refresh 命令刷新。比如下面的 SQL 语句：

```
    db2 "create table mqt testpt as (select c1 from testpt p) DATA INITIALLY
DEFERRED REFRESH DEFERRED"
    db2 "refresh table mqt testpt"
    db2 "select * from mqt testpt"
    db2 "alter table testpt p detach partition part1 into table testpt p 1"
    db2 commit
    db2 "select * from mqt_testpt"
```

这时会发现第二次查询和第一次查询的结果是相同的。

解决方法：再次手工执行刷新命令

```
    db2 "refresh table mqt testpt"
    db2 "select * from mqt_testpt"
```

3. 分区主表含有自动刷新的 MQT

问题描述：如果分区主表包含自动刷新的 MQT，那么对主表做 detach 后，MQT 和 detach 出来的独立表都不能访问。比如下面的 SQL 语句：

```
    db2 "create table mqt testpt1 as (select c1 from testpt p) DATA INITIALLY
DEFERRED REFRESH IMMEDIATE"
    db2 "REFRESH TABLE mqt_testpt"
```

```
    db2 "alter table testpt p detach partition part1 into table testpt p 1"
    SQL3601W  The statement caused one or more tables to automatically be placed
in the Set Integrity Pending state.  SQLSTATE=01586
    db2 commit
```

```
    db2 "select * from mqt testpt"

    C1
    -----------
    SQL0668N  Operation not allowed for reason code "1" on table
    "DB2INST2.MQT_TESTPT".  SQLSTATE=57016
```

```
    db2 "select * from testpt p 1"
    SQL20285N  The statement or command is not allowed while table
    "DB2INST2.TESTPT_P_1" has detached dependents.  SQLSTATE=55057
```

解决方法：对 MQT 表做约束检查，SQL 语句如下：

```
    db2 "set integrity for db2inst2.MQT_TESTPT immediate checked"
```

4. 主表或主表的分区正在被其他事务以非 UR 的隔离级别读取

问题描述：在 DB2 V9.5 中，如果主表正在被其他事务以非 UR 的隔离级别读取，那么 detach 程序会处在锁定等待状态。在 DB2 V9.7 中，如果要 detach 的分区表正在被其他事务以非 UR 的隔离级别读取，那么 detach 程序会处在锁定等待状态。比如：

```
    Open terminal 1:
    db2 +c "select * from testpt_p where c1=5 with rs"
```

```
    Open terminal 2:
    db2 "alter table testpt_p detach partition part1 into table testpt_p_1"
```

此时 terminal 2 会处在等待状态。

解决方法：提交 terminal 1 里的事务，terminal 2 会自动继续并执行成功。

2.2　多维群集(MDC)及应用案例

MDC 提供了数据在多个维上灵活、连续和自动的多维群集。它提升了查询的性能，并且减少了在插入、更新和删除期间对 REORG 和索引维护的需要。

多维群集从物理上把表数据同时沿着多个维群集起来，这与使用表上的多个独立群集的索引类似。MDC 通常用于帮助提高对大型表的复杂查询的性能。这里不需要使用 REORG 来重组群集索引，因为 MDC 会自动、动态地维护每个维上的群集。

对于 MDC，最合适的是那些具有范围、相等和连接谓词的访问多行的查询。千万不要使用具有唯一性的列作为维，因为这样会导致表不必要地变大。如果具有每种维值组合(即单元)的行不是很多，应避免使用太多的维。为获得最佳的性能，至少需要有足够的行来填满每个单元的块，也就是该表所在表空间的扩展块(extent)大小。

有了 MDC，相同的优点被扩展到多个维或聚合键上。在查询性能方面，涉及表中一个或多个指定维的范围查询将从底层的聚合获得好处。这些查询只需要访问那些包含具有指定维值的记录的页，符合条件的页将组合在一起。

随着时间的推移，当表中的可用空间被填满时，具有聚合索引的表可能变为非聚合的。然而，MDC 表可以自动、连续地在指定维上维护聚合，而不必通过重组表来恢复数据的物理顺序。

当创建 MDC 表时，会指定用于顺着它们来聚合表数据的维键。每个指定的维可以用一个或多个列来定义，这一点与索引键相同。对于每个指定的维，会自动创建维块索引，维块索引将用于快速、有效地沿着每个指定的维访问数据。此外，还会自动创建包含所有维键的块索引，块索引将用于维护插入和更新活动期间的数据聚合，以及用于对数据进行快速有效的访问。

表的维值的每一种唯一组合都形成了一个逻辑单元，逻辑单元在物理上由一些页块组成，每个页块是磁盘上的一组连续页。有一些页包含的数据在某个维块索引上具有相同键值，包含这些页的一组块被称作切片(slice)。表的每个页只存储在一个块中，表的所有块由相同数量的页组成，即所谓的分块因子(blocking factor)。分块因子与表空间的盘区大小相等，因此块边界与盘区边界呈线性关系。

2.2.1　创建 MDC 表

为了创建 MDC 表，需要使用 ORGANIZE BY 参数指定表的维，如下所示：

```
CREATE TABLE MDCTABLE(
Year INT,
Nation CHAR(25),
Colour VARCHAR(10),
... )
ORGANIZE BY(Year, Nation, Color)
```

在上面的代码中，新建的表将按 year、nation 和 color 这几个维来组织，逻辑上看起来如图 2-3 所示。

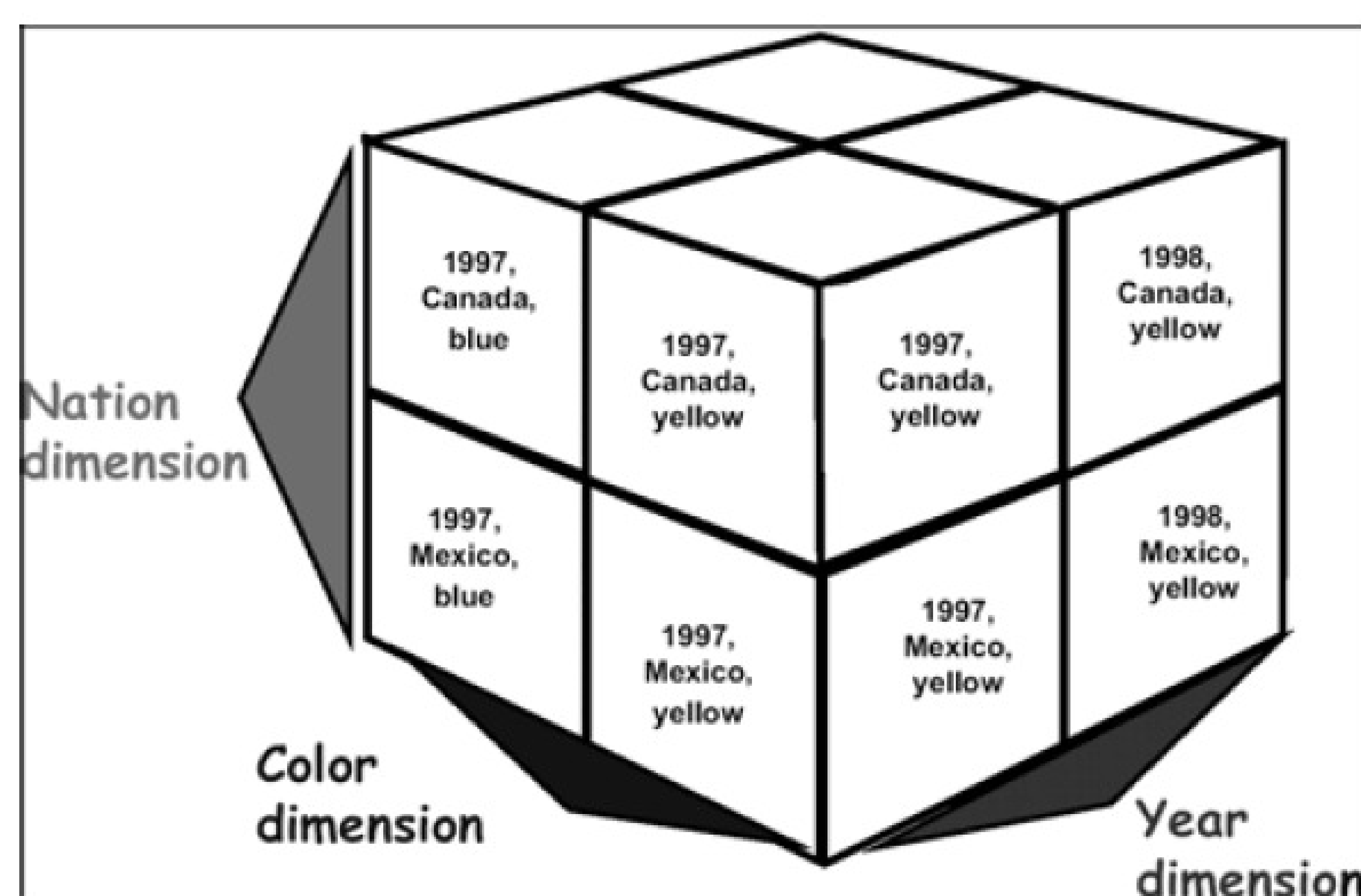


图 2-3　逻辑上的效果

由于不能将表改成 MDC 表，因此在创建数据库之前，您应该尽可能地了解业务特点和业务需求，看看是应该创建 MDC 表还是创建普通的表。

2.2.2　MDC 测试案例

MDC 可以大大提高查询的速度，但是装载和插入操作会受到一定的影响，下面是一个实际生产中的例子。

测试 MDC 的过程说明：

(1) 用交易流水表 PB_CSTLOG(比较大的表)进行测试，建立 PB_CSTLOG_TEST 表，结构和 PB_CSTLOG 完全一样。

```
CREATE TABLE PB_CSTLOG_TEST like PB_CSTLOG;
```

(2) 首先在没有建立 MDC 索引的情况下将 PB_CSTLOG 的数据导入到 PB_CSTLOG_TEST 中，记录装载时间 time1。

```
insert into PB_CSTLOG_TEST select  * from PB_CSTLOG
```

(3) 将表PB_CSTLOG_TEST的关于数据网点号TRANDATE= '2012-05-01'的记录删除,记录时间 time2。

```
delete from PB_CSTLOG_TEST where TRANDATE='2012-05-01'
```

(4) 然后建立 PB_CSTLOG_TEST1 表，结构和 PB_CSTLOG 一样，只是用日志时间(TRANDATE)建立 MDC 索引。

```
create table PB_CSTLOG_TEST1(...) ORGANIZE BY DIMENSIONS (TRANDATE)
```

(5) 再将 PB_CSTLOG 的数据导入到 PB_CSTLOG_TEST1 中，记录下装载时间 time3。

```
insert into PB_CSTLOG_TEST1 select  * from PB_CSTLOG
```

(6) 将表 PB_CSTLOG_TEST1 的关于数据网点号 TRANDATE= '2012-05-01'的记录删除，记录下时间 time4。

```
delete from PB_CSTLOG_TEST1 where TRANDATE='2012-05-01'
```

(7) 然后再建立 PB_CSTLOG_TEST2 表，结构和 PB_CSTLOG 一样，只是用日志时间(TRANDATE)建立普通索引。

```
create table PB CSTLOG TEST2(...)
create index IDX_PB_CSTLOG_TEST2 ON PB_CSTLOG_TEST2(TRANDATE)
```

(8) 再将 PB_CSTLOG 的数据导入到 PB_CSTLOG_TEST2 中，记录下装载时间 time5。

```
insert into PB_CSTLOG_TEST2 select  * from PB_CSTLOG
```

(9) 将表 PB_CSTLOG_TEST2 的关于数据网点号 TRANDATE= '2012-05-01'的记录删除，记录下时间 time6。

```
delete from PB_CSTLOG_TEST2 where TRANDATE='2012-05-01'
```

(10) 比较 time1、time3 和 time5，time2、time4 和 time6。

详细测试记录见表 2-4。

表 2-4　详细测试记录

测试项目	记录数	字段数	SQL	消耗时间
MDC 装载效率	3824432	11	见 MDC 测试过程	02:04.0
MDC 查询效率	83624	11	见 MDC 测试过程	0.641
普通 IDX 装载效率	3824432	11	见 MDC 测试过程	01:45.0
普通 IDX 查询效率	83624	11	见 MDC 测试过程	2.219

测试结论：

- 未建立 MDC 索引表时，装载时间 time1=1.33.4；建立 MDC 索引后，表装载时间 time3=2:04
- 未建立 MDC 索引表时，查询时间 time2=15.281；建立 MDC 索引后，表查询时间 time4=0.641

普通索引装载时间为 time5=1:45，查询时间为 time6=2.219，由此看来普通索引比 MDC 索引的装载时间要快，而查询时间要慢一些

在有数据的情况下：

- 普通索引装载时间为 4:11.438
- MDC 索引装载时间为 8:11.563

由此可见，有 MDC 索引的加载时间要大于普通加载时间

2.2.3 MDC 考虑

以下总结了在设计 MDC 表时要考虑的一些方面：

- 在确定候选维的时候，应寻找那些不是太细粒度的属性，以允许在每个单元中存储更多的行。这种方法能更好地使用块级索引。
- 更高的数据量能提高填充密度。
- 如果只是用于分析，那么首先将数据装载为非 MDC 会比较有用。
- 表空间的扩展数据块大小对于有效的空间使用是一个关键的参数。
- 虽然 MDC 表可能要求一开始对数据有更多的理解，但是获得的回报是查询时间有望大幅度缩短。
- 有些数据可能不适合用 MDC 表，使用标准聚合索引可能更好。
- 虽然更小的扩展数据块大小可以提高对磁盘空间的使用效率，但是还应该考虑查询的 I/O。
- 更大的扩展数据块大小通常会减少 I/O 成本，因为每次可以读取更多的数据。反过来，这样又会导致更小的维块索引，因为每个维值将需要更少的块。而且插入操作将变得更快，因为对新块的需要更少了。

2.3 表分区和多维集群表的使用

在同时是多维集群表和数据分区表的表中，可以同时在表分区的范围分区规范和多维集群(MDC)键中使用列。与只单独使用多维集群或分区功能相比，同时是多维集群表和分区表的表可以获取较详细的数据分区和块消除。在许多应用中将 MDC 键列指定为不同于

对表进行分区的列很有用。应该注意的是，表分区是多列的，而 MDC 是多维的。

典型仓库事实表可能采用以下设计：

- 在 Month 列中创建数据分区。
- 为转出的每个时间段(例如 1 个月和 3 个月)定义数据分区。
- 在 Day 和 1 到 4 个其他维的基础上创建 MDC 维。典型的维有生产线和区域。
- 所有数据分区和 MDC 集群都分布在所有数据库分区中。

MDC 和表分区具有一些相同的好处。表 2-5 列示了组织中的潜在需求并根据先前确定的特征确定建议的组织方案。

表 2-5　潜在需求、方案及建议

问　　题	方　　案	建　　议
转出期间的数据可用性	表分区	使用 DETACH PARTITION 子句来转出大量数据，并且只出现最少中断
查询性能	表分区和 MDC	MDC 最适合用来查询多个维。表分区通过数据分区消除提高性能
最少重组	MDC	MDC 维护集群，从而减少进行重组的必要性
在传统脱机窗口中转出一个月或更长时间的数据	表分区	数据分区可以完全解决此需求。MDC 不起任何作用，并且仅 MDC 并不适合
在短时间脱机窗口(小于 1 分钟)期间转出一个月或更长时间的数据	表分区	数据分区可以完全解决此需求。MDC 不起任何作用，并且仅 MDC 并不适合
转出一个月或更长时间的数据，并同时在不损失任何服务的情况下使表对于提交查询的企业用户完全可用	MDC	MDC 只能解决一部分需求。由于表处于脱机状态的时间段太短，表分区并不适合
每天装入数据(ALLOW READ ACCESS 或 ALLOW NO ACCESS)	表分区和 MDC	此时 MDC 具有很多好处，表分区具有递增的好处
“连续”装入数据(ALLOW READ ACCESS)	表分区和 MDC	此时 MDC 具有很多好处，表分区具有递增的好处
“传统 BI”查询的查询执行性能	表分区和 MDC	MDC 特别适合用来查询立方体/多个维。表分区通过分区消除提高性能

(续表)

问　　题	方　　案	建　　议
通过消除进行重组的必要性或减少执行任务所产生的不良影响，使重组所带来的不良影响降到最低	MDC	MDC 维护集群，从而减少进行重组的必要性。如果使用 MDC，那么数据分区不提供递增好处。但是，如果不使用 MDC，那么表分区通过在分区级别维护一些粗粒度集群会有助于减少重组的必要性

考虑一个具有键列 YearAndMonth 和 Province 的表。一种合理的规划此表的方法是按日期进行分区，每两个月添加一个数据分区。此外，还可以按 Province 进行组织，以便任何两个月日期范围内的特定省份的所有行集群在一起，如图 2-4 所示。

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (Province);
```

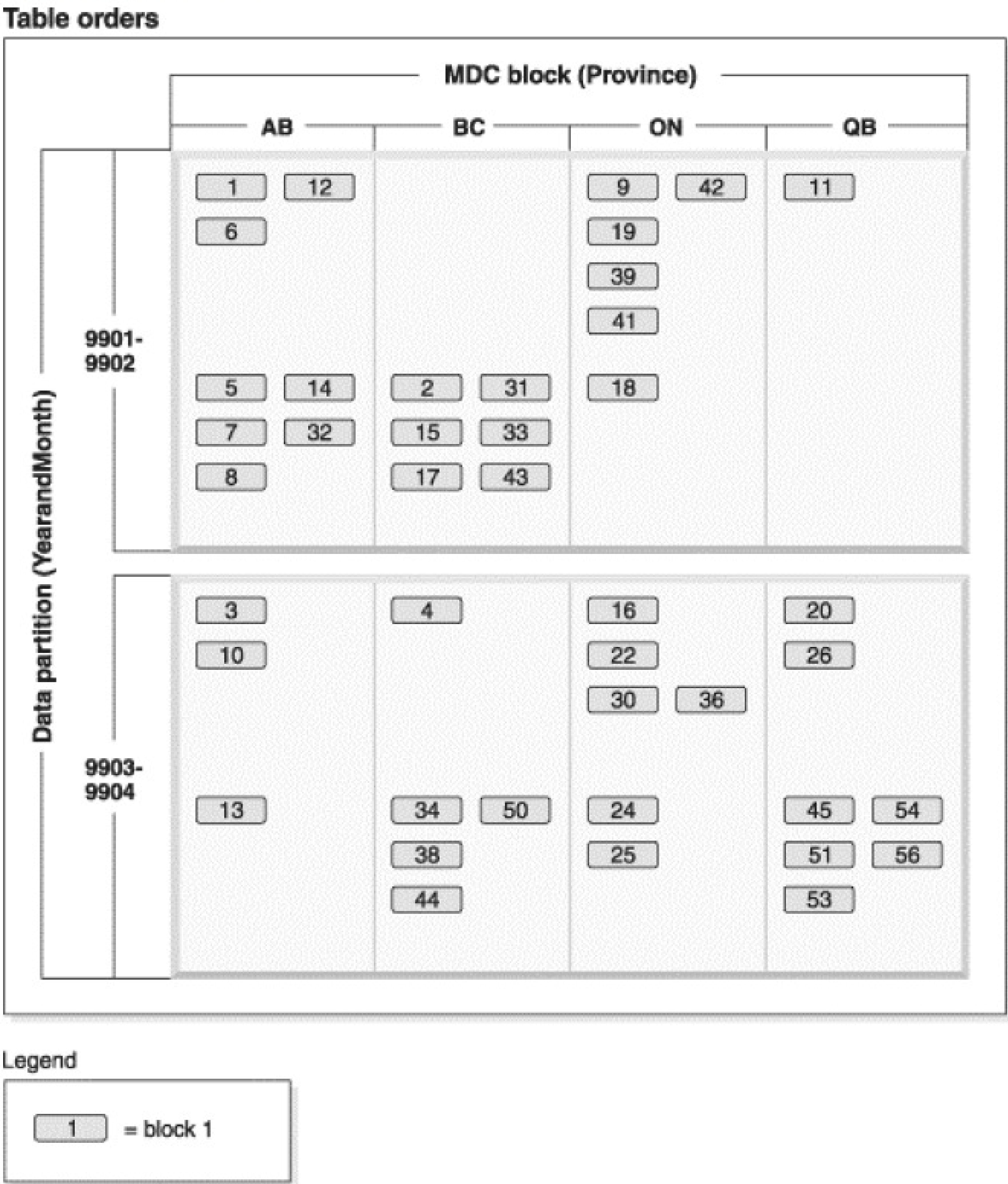


图 2-4　按 YearAndMonth 分区并按 Province 组织的表

通过将 YearAndMonth 添加至 ORGANIZE BY 子句，可以获得较高的详细程度，如图 2-5 所示。

```
CREATE TABLE orders (YearAndMonth INT, Province CHAR(2))
PARTITION BY RANGE (YearAndMonth)
(STARTING 9901 ENDING 9904 EVERY 2)
ORGANIZE BY (YearAndMonth, Province);
```

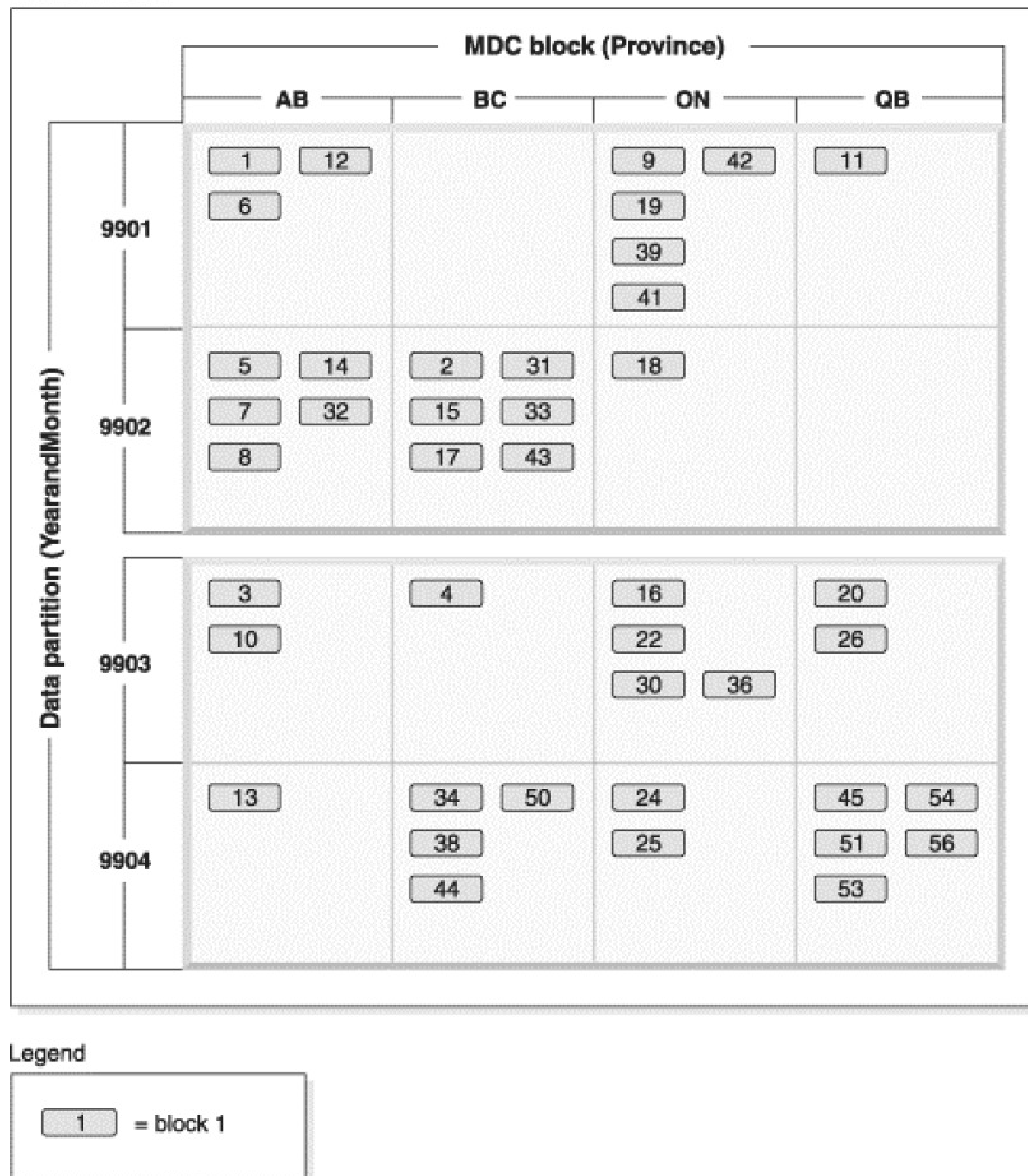


图 2-5　按 YearAndMonth 分区并按 Province 和 YearAndMonth 组织的表

如果分区导致每个范围内只有单个值，那么通过在 MDC 键中包括表分区列不能获得任何好处。

注意事项

- 与基本表相比，MDC 表和分区表都需要一些存储器。这些存储器需求是附加的，但相对于能够带来的好处来说，它们是合理的。
- 如果选择不在分区数据库环境中组合表分区和 MDC 功能，那么在您可以非常自信地预计数据分发情况时(通常也就是此处讨论的系统类型情况)，使用表分区是最好的选择。否则，应考虑 MDC。
- 对于使用 DB2 V9.7 修订包 1 或更高发行版创建的数据分区 MDC 表，表的 MDC 块索引是分区索引。对于使用 DB2 V9.7 或更早发行版创建的数据分区 MDC 表，表的 MDC 块索引是非分区索引。

2.4 物化查询表及应用案例

2.4.1 物化查询表(MQT)

MQT 可用于提升使用 GROUP BY、GROUPING、RANK 或 ROLLUP OLAP 函数的查询性能。MQT 的使用对用户来说是透明的，DB2 会选择何时使用 MQT 来达到优化的目的。DB2 使用 MQT 在内部维护被查询的分组的汇总结果，这样用户就可以直接访问 DB2 维护的分组，而不必去读动辄数 GB 的数据来寻找答案。这些 MQT 还可以在分区间复制，以避免这种信息在分区间散播，从而帮助提升合并连接(collocated join)的性能。

MQT 是一种以一次查询的结果为基础定义的表。包含在 MQT 中的数据来自定义 MQT 时基于的一个或多个表。

可以将 MQT 看作一种物化的视图。视图和 MQT 都是基于查询来定义的。每当视图被引用时，视图基于的查询便会运行。但是，MQT 实际上是将查询结果保存为数据，您可以使用 MQT 中的这些数据，而不是使用底层表中的数据。

MQT 可以显著提高查询的性能，尤其是提高复杂查询的性能。如果优化器确定查询或查询的一部分可以用 MQT 来解决，那么就会重写查询，以便利用 MQT。

MQT 可以在创建表时定义，或者定义为系统维护的 MQT，或者定义为用户维护的 MQT。

1. 系统维护的 MQT

这种物化查询表中的数据是由系统维护的。当创建这种类型的 MQT 时，可以指定表数据是 REFRESH IMMEDIATE 还是 REFRESH DEFERRED。通过 REFRESH 关键字可以指定如何维护数据。DEFERRED 的意思是，表中的数据可以在任何时候通过 REFRESH TABLE 语句来刷新。不管是 REFRESH DEFERRED 还是 REFRESH IMMEDIATE 类型的系统维护的 MQT，对它们的 insert、update 或 delete 操作都是不允许的。但是，对于

REFRESH IMMEDIATE 类型的系统维护的 MQT，可以通过对底层表的更改(insert、update 或 delete 操作)来更新。

例如，创建由系统维护的 MQT：

```
connect to sample
...
alter table employee add unique (empno)
alter table department add unique (deptno)
create table emp summary as (select e.empno, e.firstnme, e.lastname,
e.phoneno, d.deptno, substr(d.deptname, 1, 12) as department, d.mgrno from
employee e, department d
  where e.workdept = d.deptno)
   data initially deferred refresh immediate
set integrity for emp_summary immediate checked not incremental
select * from emp summary
EMPNO  FIRSTNME     LASTNAME        PHONENO DEPTNO DEPARTMENT   MGRNO
------ ------------ --------------- ------- ------ ------------ ------
000010 CHRISTINE    HAAS            3978    A00    SPIFFY COMPU 000010
000020 MICHAEL      THOMPSON        3476    B01    PLANNING     000020
000030 SALLY        KWAN            4738    C01    INFORMATION  000030
000050 JOHN         GEYER           6789    E01    SUPPORT SERV 000050
000060 IRVING       STERN           6423    D11    MANUFACTURIN 000060
000070 EVA          PULASKI         7831    D21    ADMINISTRATI 000070
000090 EILEEN       HENDERSON       5498    E11    OPERATIONS   000090
000100 THEODORE     SPENSER         0972    E21    SOFTWARE SUP 000100
000110 VINCENZO     LUCCHESSI       3490    A00    SPIFFY COMPU 000010
000120 SEAN         O'CONNELL       2167    A00    SPIFFY COMPU 000010
000130 DOLORES      QUINTANA        4578    C01    INFORMATION  000030
...
000340 JASON        GOUNOT          5698    E21    SOFTWARE SUP 000100
  32 record(s) selected.
connect reset
```

上面的代码展示了如何创建 REFRESH IMMEDIATE 类型的系统维护的 MQT。这个表名为 EMP_SUMMARY，它基于 SAMPLE 数据库中的底层表 EMPLOYEE 和 DEPARTMENT。由于 REFRESH IMMEDIATE MQT 要求查询的 select 列表中引用的每个表中至少有一个唯一键，因此我们首先在 EMPLOYEE 表的 EMPNO 列上定义了唯一性约束，另外还在 DEPARTMENT 表的 DEPTNO 列上也定义了唯一性约束。DATA INITIALLY DEFERRED 子句的意思是，在执行 CREATE TABLE 语句的时候，并不将数据插入到表中。MQT 被创建好之后，便处于检查暂挂(check pending)状态，在对它执行 SET INTEGRITY 语句之前，不能查询它。IMMEDIATE CHECKED 子句规定，根据用于定义该 MQT 的查询对数据进

行检查并刷新数据。NOT INCREMENTAL 子句规定对整个表进行完整性检查。通过查询 EMP_SUMMARY 物化查询表，发现它现在已经填入了数据。

2. 用户维护的 MQT

这种物化查询表中的数据是由用户维护的。只有 REFRESH DEFERRED 物化查询表可以定义为 MAINTAINED BY USER。不能对用户维护的 MQT 发出 REFRESH TABLE 语句(用于系统维护的 MQT)。但是，用户维护的 MQT 却允许对它们执行 insert、update 或 delete 操作。

例如，创建由用户维护的 MQT：

```
connect to sample
...
create table sum 2005 report as (select distinct e.empno, e.firstnme,
 e.lastname, e.workdept, e.phoneno, 'Ontario' as region,
  year(s.sales date) as year from employee e, sales s
   where e.lastname = s.sales person and year(s.sales date) = 1995
    and left(s.region, 3) = 'Ont')
     data initially deferred refresh deferred maintained by user
set integrity for sum 2005 report materialized query immediate
 unchecked
export to sum 2005 report.del of del
 select distinct e.empno, e.firstnme, e.lastname, e.workdept, e.phoneno,
  'Ontario' as region, year(s.sales date) as year from employee e,
  sales s
   where e.lastname = s.sales person and year(s.sales date) = 1995
    and left(s.region, 3) = 'Ont'
...
Number of rows exported: 2
import from sum 2005 report.del of del insert into sum 2005 report
...
Number of rows committed    = 2
insert into sum 2005 report
 values ('006900', 'RUSS', 'DYERS', 'D44', '1234', 'Ontario', 1995)
select * from sum 2005 report
EMPNO  FIRSTNME      LASTNAME       WORKDEPT PHONENO REGION   YEAR
------ ------------ -------------- -------- ------- ------- -----------
000110 VINCENZO      LUCCHESSI      A00      3490    Ontario  1995
000330 WING          LEE            E21      2103    Ontario  1995
006900 RUSS          DYERS          D44      1234    Ontario  1995
  3 record(s) selected.
connect reset
```

上述代码展示了如何创建 REFRESH DEFERRED 类型的用户维护的 MQT。这个表名为 SUM_2005_REPORT，它基于数据库 SAMPLE 中的底层表 EMPLOYEE 和 SALES。同样，DATA INITIALLY DEFERRED 子句的意思是，在执行 CREATE TABLE 语句的时候，并不将数据插入到表中。MQT 被创建之后，便处于检查暂挂状态，在对它执行 SET INTEGRITY 语句之前，不能查询它。MATERIALIZED QUERY IMMEDIATE UNCHECKED 子句规定，该表将启用完整性检查，但是不必检查它是否违反了完整性约束，便可以使之脱离检查暂挂状态。

接下来，为了填充数据到 MQT 中，我们将导入从 EMPLOYEE 和 SALES 表中导出的数据。用于导出数据的查询与用于定义 MQT 的查询是一致的。然后，我们将插入另外一条记录到 SUM_2005_REPORT 表中。

通过查询 SUM_2005_REPORT 物化查询表发现，它现在已经填入了刚才导入和插入的数据，这表明用户维护的 MQT 的确可以直接被修改。

2.4.2　MQT 总结

在 SYSCAT.TABDEP 系统编目视图中，对于 MQT 在其他对象上的每个依赖关系，都有相应的一行。可以查询这个视图，以获得我们创建的 MQT 的依赖关系的总结。MQT 有一个值为'S'的 DTYPE。TABNAME 列显示 MQT 的名称，BNAME 列显示相应的 MQT 所依赖的数据库对象的名称。BTYPE 列显示对象类型：'T'表示表，'I'表示索引，'F'表示函数实例。

例如，查询 SYSCAT.TABDEP 系统编目视图，查看 MQT 在其他数据库对象上的依赖关系：

```
    connect to sample
    ...
    select substr(tabname,1,24) as tabname, dtype, substr(bname,1,24) as bname,
btype from syscat.tabdep where tabschema = 'MELNYK' and dtype = 'S'
    TABNAME                    DTYPE  BNAME                       BTYPE
    ---------------------- -----  ------------------------  -----
    EMP                         S      DEPARTMENT                  T
    EMP                         S      EMPLOYEE                    T
    EMP                         S      SQL050829104058970          I
    EMP                         S      SQL050829104058800          I
    EMP_SUMMARY                 S      EMPLOYEE                    T
    SUM_2005_REPORT             S      LEFT1                       F
    SUM_2005_REPORT             S       SALES                      T
    SUM_2005_REPORT             S      EMPLOYEE                    T
    SALES_SUMMARY               S      SALES                       T
```

```
  9 record(s) selected.
connect reset
```

我们看到了一个物化查询表，其定义基于某个查询的结果，可以将它看作一种物化视图。MQT 很重要，因为它们可以显著减少复杂查询的响应时间，对于做 T+1 之类的报表特别合适。

2.5 MDC、数据库分区、MQT 和表分区配合使用

在数据仓库中，事实表或历史表的大小是摆在设计人员和管理员面前的一个挑战。这些表通常包含数亿行数据，有时候甚至包含数千亿行数据。对于这种规模的表，主要关心以下几点：

- 查询性能
- 将大量新数据插入到这些表中
- 每月或每个季度删除大量过时的数据

对于这样的特殊查询，除了单独用到我们上面所讲的表分区、数据库分区、MQT 和 MDC 外，有时候还需要这几种技术相互配合以提供更高的性能，因为这几种技术可以相互补充。

1. 三个互补的 CREATE TABLE 选项

CREATE TABLE 语句现在提供了 3 种方式来组织数据库表中的数据，如表 2-6 所示。

表 2-6　使用 CREATE TABLE 语句

CREATE TABLE 语句中的子句	DB2 特性名称
DISTRIBUTE BY HASH	DPF——数据库分区特性
ORGANIZE BY DIMENSION	MDC——多维集群
PARTITION BY RANGE	TP——表分区

可以任意组合、使用这些子句，以达到期望的效果。表 2-7 总结了与这些特性相关的术语，此处用到的其他一些特性也包含在内。

表 2-7　DB2 特性术语

DB2 特性名称	一部分的名称	用于分区数据的列	其他术语
数据分区特性(Data Partitioning Feature，DPF)	数据库分区	分布键(distribution key)	在之前的版本中，分布键被称作分区键
多维集群(MultiDimensional Clustering，MDC)	单元格，由一些块组成	维	块索引
表分区(TP)	数据分区	表分区键	

2. 三种技术对比

每种特性都为分组表中的数据提供了独特的方法，并对解决与事实表或历史表相关的需求有独特的作用。

DPF 是最老的特性，通过它可以将数据库分成多个数据库分区。每个数据库分区有它自己的一组计算资源，包括 CPU 和存储。在 DPF 环境中，根据 CREATE TABLE 语句中指定的分区键，表中的每个行被分布到一个分区上。当处理查询时，请求也相应被划分成多个部分，以便让各个数据库分区各自处理其负责的那些行。实际上，DPF 是一种可伸缩特性。DPF 可以通过增加数据库分区来提高处理能力。因此，随着表的增长，仍然可以保持较高的查询性能。这种能力常常被称作使用 DB2 的无共享架构提供线性的可伸缩性。

DPF 的作用不仅仅体现在表设计上，它还是调整和配置整个数据库系统的一种方法。现在已经有了关于配置那样的系统以取得最佳性能、可靠性和增长能力的建议实践。客户可以购买建议的硬件和软件以及配置，它们都放在称作 BCU(Balanced Configuration Unit)的解决方案中。

MDC 是在 DB2 V8 中引入的，通过它可以在物理上将在多个维上具有类似值的行聚合在一起放到磁盘上。这种聚合能为常见分析性查询提供高效的 I/O。例如，对于 VCITEM(科目)='71005'，VCBRNO(分行号)='厦门分行'，并且 VCTRDT(交易日期) ='2012/09'的所有行，可以将它们存储在相同的存储位置，即所谓的块(block)。在 CREATE TABLE 语句中定义维的时候，就为每种值的组合预留了存储空间。实际上，MDC 是一个能最大化查询性能的特性，对于数据仓库中常用的查询更是如此。这包括需要根据几个列中的值的组合选择行的查询。例如：

```
    VCTRDT(交易日期)is between "2012/09/01" and "2012/09/31" AND VCITEM (科目)=
'71005'AND VCBRNO (分行号)= '厦门分行'
```

TP 是在 DB2 V9 中引入的，与 MDC 类似，它也可以将具有近似值的行存储在一起。

但是，TP 的以下特征是 MDC 所不具备的：

- TP 支持按照维将表分区成多个数据分区。一种常见的设计是为每个月的数据创建数据分区。MDC 则支持定义多个维。
- 通过 TP，用户可以手动地定义每个数据分区，包括将被包括到那个分区的值的范围。MDC 则为每种唯一的 MDC 维值组合自动定义一个单元格(并创建块来存储那个单元格的数据)。
- 每个 TP 分区是单独的数据库对象(不同于其他作为单个数据库对象的表)。因此，TP 支持为 TP 表附加(attach)和卸除(detach)数据分区。卸除的分区成为常规表，而且必要时可以将每个数据分区放在它自己的表空间中。

实际上，TP 相对其他特性的优势在于为表添加或删除大量数据这个方面，即转入(roll-in)和转出(roll-out)。对于熟悉使用 Union All View(UAV)来按日期对历史表分区的读者，TP 可以作为一种功能相似但是更为高级的解决方案。表 2-8 对 DPF、MDC、TP 的特性做了简要对比。

表 2-8　DPF、MDC 和 TP 特性简要对比

特　　性	如何组织数据	优　　点
DPF	将行均匀地分布在多个数据库分区上	可伸缩性——随着数据库的增长增加计算资源(也就是数据库分区)，用于高性能计算
MDC	将在多个维上具有近似值的行放在表中相同的物理位置，即所谓的块	查询性能——组织数据的方式有利于获得更快的检索速度，对于由多个谓词指定范围的查询尤其有效
TP	将所有行放在同一数据分区的一个指定范围的维中	数据移动——通过添加或删除整个数据分区，可以增加或删除大量数据

3. 技术的互补特性

这 3 种 DB2 的高级设计技术既是独立的，又是互补的。当设计表时，对每个特性的使用可以认为是独立的。例如：

- 是否使用 MDC 和 TP 对于决定 DPF 的分布键没有影响。
- 一个列是否被用作 MDC 维对于是否应该将它作为表分区键没有影响，反之亦然。每个决定都可以单独做出。

每个特性的工作方式，例如索引方面，不会随着新的分区特性的引入而改变。例如，

当引入 MDC 时，它的索引方面不会改变 DPF 在索引方面的工作方式。同样，当引入 TP 时，也不会改变 DPF 或 MDC 在索引方面的行为。在学习这些特性的时候，记住这一点有助于避免陷入困惑。

通常来讲，一个特性不能用于解决数据库设计中与另外一个特性相关的不足或问题。值得注意的例子有：

- TP 不能解决与 DPF 相关的问题，不管这个问题是 DPF 数据倾斜还是 DPF 中的管理活动速度变慢。不管有没有同时使用 TP，DPF 之前的补救方法仍然适用。
- TP 不能用于纠正不好的 MDC 设计。

总之，如果存在与 DPF、MDC 或 TP 相关的问题，那么还是应该尝试适用于那个特性的解决方法。

4. 表设计

数据仓库中的事实表或历史表非常适合使用上述每种特性，如表 2-9 所示。

表 2-9　事实表拥有适合使用 DB2 分区特性的特征

特　　性	适合的表的特征	事实表的特征
DPF	大型表——大到无法仅依靠单独一组 CPU 和 I/O 通道来处理	事实表是最大的数据库表。它们常常包含数亿行数据，有时候甚至包含数千亿行数据
MDC	结果集返回在多个维上具有近似值的行的查询	事实表(以及通常所说的数据仓库)是为支持这种类型的查询而设计的
TP	周期性地添加大量数据，然后在数据到期后又删除大量数据	在事实表中，常常是每天都添加新数据。通常每月或每个季度都要删除过时的数据

1) 表设计的经验法则

对于 DPF，在选择分布键时，应首选那种能使数据行均匀地分布在多个数据库分区上的列。当不具备这样的条件时，就会造成数据倾斜。这意味着一个或一些数据库分区承担了更重比例的表行，从而造成了性能瓶颈。具有很多不同值的列是较好的选择。还有一种考虑是选择能最大化连接性能的列。

另一个 DPF 设计决定是数据库分区的数量。数据库分区的数量不算是表设计方面的考虑。实际上，它是整个系统设计上的考虑，需要根据整个数据库预期的原始数据大小和服务器硬件的能力来决定。很多系统需要的数据库分区不超过 20 个。但是，最大的系统可能需要更多的数据库分区。由于数据仓库有越来越大的趋势，数据库分区的数量有望增加。

对于 MDC，一个关键的决定是用哪些列作为 MDC 维。设计上的挑战是根据业务需求

找到最佳的一组维和最佳的粒度，使得组织最大化，存储需求最小化。较好的选择是具有以下一部分或全部特征的列：

- 用于范围、等于或 IN 列表谓词
- 用于转入、转出或其他大规模的行删除
- 被 GROUP BY 或 ORDER BY 子句引用
- 外键列
- 星型数据库的事实表中 JOIN 子句中的列
- 粗粒度，也就是说不同的值很少的列

典型的设计是用表示日期的列作为 MDC 维，再加上 0 到 3 个其他列作为其他维，例如 VCITEM(科目)= '71005'，VCBRNO(分行机构号)= '厦门分行'。

对于 TP，设计决定包括选择用作表分区键的列和分区的数量。通常表分区键是基于时间的列。每个分区与每次转入的数据量相符。例如，每月转出数据的表，对于每个月的数据都有一个分区。对于 TP，在设计时需要特别考虑的一点是，需要处理不在 CREATE TABLE 语句中定义的值范围内的行。

对于需要每月根据 VCTRDT(交易日期)转出(roll-out)数据的数据库，一种典型的设计是使用 VCTRDT(交易日期)作为表分区键，并为每个月创建单独的分区。

通常，有一个 MDC 维是基于时间的列，这样一来，同一列既可以用于 MDC，又可以用于 TP。MDC 的粒度可以比 TP 数据库分区更细一些。

表 2-10 总结了以上内容。

表 2-10　设计方面的经验法则总结

分区特性设计决定	经 验 法 则
DPF——用作分布键的列	首选是具有很多不同值的列
MDC——用作 MDC 维的列	一种典型的设计是选择一个表示日期的列，再加上 0 到 3 个其他列，例如 VCITEM(科目)和 VCBRNO(分行机构号)
TP——用作表分区键的列和分区的数量	选择一个基于时间的列，定义与每次转出的数据量相符的分区

2) 设计的例子

表 2-11 显示了一些典型的表设计的例子。Cmvca 历史传票表代表关系数据仓库中的一个典型的表。Recent Cmvca 历史传票表代表运营数据存储中的一个表，这种数据存储实际上是只有最近数据的数据仓库。

表 2-11　表设计的例子

分区属性	Cmvca 历史传票表	Recent Cmvca 历史传票表
DPF——用作分布键的列	不采用数据库分区	不采用数据库分区
DPF——分区数量	-	-
MDC——用作维的列	VCTRDT(交易日期 Year+Month)=36 月 VCITEM(科目)=112 VCBRNO(分行机构号)=30	VCTRDT(交易日期，Days)=90 日 VCITEM(科目)=112 VCBRNO(分行机构号)=30
TP——用作表分区键的列和分区的数量	VCTRDT(交易日期)，每个月一个分区	VCTRDT(交易日期)，每个月一个分区
# of rows(1 million per day)	8 亿条记录	7000 万
# of columns	30	30
# of indexes	4	15

注：对于 Recent Cmvca 历史传票表上的 MDC 维，另一种设计方案是以更细的粒度定义 VCTRDT(交易日期)，例如每周或每天。更好的查询性能与增加的存储需求之间的平衡取决于数据和查询的特征。

5. MQT

MQT 和分区特性都是为在相同的情况下(数据仓库事实表或历史表)使用而设计的。因此，为了全面考虑如何使用此处提到的分区特性，需要解决涉及 MQT 情况下的一些特殊考虑。

MQT 是基于查询结果而定义的表。从另一种角度来看，MQT 就像结果集存储在表中的视图。MQT 可以减少涉及以下方面的复杂查询的响应时间：

- 基本表中针对数据的聚类(avg、sum、count、max 和 min)或计算
- 基本表的连接
- 一个或多个较大基本表中常被访问的一部分数据

当基本表中的数据发生改变时，需要相应地更新 MQT。MQT 特性为适应各种运营需求而进行更新提供了多种选项。

1) 与分区特性的比较(参见表 2-12)

表 2-12 MDC、DPF、TP 和 MQT 特性简要对比

特　性	如何组织数据	优　点
DPF	将行均匀地分布在多个数据库分区上	可伸缩性——随着数据库的增长增加计算资源(也就是数据库分区)
MDC	将在多个维上具有近似值的行放在表中相同的物理位置，即所谓的块	查询性能——组织数据的方式有利于获得更快的检索速度，对于由多个谓词指定范围的查询尤其有效
TP	将所有行放在同一数据分区的一个指定范围的维中	数据移动——通过添加或删除整个数据分区，可以增加或删除大量数据
MQT	将查询的结果存储在表中	查询性能——对于涉及较高代价的操作，例如复杂的连接和表扫描的查询，预先计算结果集并存储(物化)结果集

2) 与分区特性一起设计 MQT

与分区特性一起设计 MQT 时，在设计上需要考虑以下几点：

- 可以在使用分区特性的任意组合的表上创建 MQT。例如，可以在一个或多个使用 MDC 和 TP 的表上创建 MQT。基本表上分区特性的使用不需要考虑将来是否会在这个表上创建 MQT。然而，MQT 的设计却可能受到基本表上使用的分区特性的影响。例如，如果基本表使用 DPF 进行分区，那么 MQT 的设计就应该考虑是否要在各个数据库分区上复制 MQT。
- MQT 也可以使用分区特性。例如，可以使用 MDC 或 TP 对 MQT 分区。

3) 典型的设计

接下来进一步介绍前面的 Recent Cmvca 历史传票表的例子，下面是这些表上定义的一些 MQT：

- MQT 1——每个分行机构每种科目每天的交易汇总
- MQT 2——每个分行机构每天每月每年的交易汇总

6. 关键考虑：查询性能

1) 对这一考虑的描述

现在我们来看看在评价和使用这些特性时关键的考虑角度：性能，尤其是通常的数据仓库业务的用户查询的性能。这些查询具有以下特征：

- 从事实表中选择在几个维上符合标准的行，这意味着需要将几个维表与事实表相连接。

- 使用分组或聚类函数，例如 COUNT、GROUP BY 和 ORDER BY。
- 返回包括很多行的结果集，从数千行到数百万行。
- 这些查询是由用户或他们的 BI 工具生成的。这意味着这些查询在更多情况下是临时性的，事务处理系统中的性能测试和调优方法对于它们并不适合。

虽然人们倾向于获得更快的性能，但最好还是说成更好的性能。谈到性能，就应该包括以下一些方面：

- 峰值查询执行性能。
- 查询执行性能的稳定性。
- 对于数据仓库中各种具有不同特征的工作负载的性能。
- 设计数据库以达到性能目标是否容易。
- 为达到性能目标需要付出的代价。

与过去相比，现在在设计数据库以达到性能目标的过程中还需要考虑的一点是硬件的发展趋势。随着 CPU 处理能力的不断提升以及存储设备容量的不断扩大，I/O 带宽是一个潜在的性能瓶颈。在这种环境下，I/O 效率是设计时要关键考虑的一点。

上面我们讲解了每种设计技术(MQT、DPF、TP 和 MDC)对查询执行性能的作用。下面我们讲解这些技术如何影响转入和转出的性能。

2) DB2 分区特性发挥的作用

使用 DPF 可以提供更多的计算资源，从而对性能产生积极的影响。当 DB2 优化器为查询形成查询访问计划时，它将工作划分到多个数据库分区上，这些数据库分区是并行工作的。之后，再收集各个数据库分区上得到的结果，并返回给查询提交者。

MDC 对性能的贡献在于提高检索数据的效率。在多个维上具有近似值的数据存储在相同的位置，这使得 I/O 操作变得更高效，而 I/O 操作正是数据仓库中一个常见的瓶颈。而且，MDC 的特性还包括块索引。在块索引中，对于每个块的数据(而不是每一行的数据)都有一个条目，这使得执行索引操作时有更高的效率。为了发挥它潜在的性能，必须为 MDC 表设计一组最佳的(或者至少是够好的)维。MDC 只对那些包括维列的查询有好处。MDC 对于查询是完全透明的。最后，MDC 在管理方面有两个值得注意的优点：

- MDC 块索引意味着需要的 RID 索引更少。管理方面的一个优点是用于索引的存储空间减少了。
- 由于新行是插在表中具有近似值的行附近的位置，因此数据仍然是聚合的，而不需要运行 REORG 实用程序。

TP 通过分区排除来提高查询性能。例如，假设 Cmvca 历史传票表有 36 个分区，每个月对应一个分区。对于一个选择过去 12 个月的数据的查询，优化器知道不必扫描这 12

个月之外的其他分区的数据。这种分区排除既适用于索引扫描，也适用于表扫描。TP 只对那些包括表分区键列的查询有利。

7. 关键考虑：插入新数据

1) 对这一考虑的描述

将来自运营系统的新数据插入到数据仓库中的事实表，这个过程称作 ETL、摄取(ingest)、填充数据仓库或转入。下面的例子演示了可能遇到的各种情况。

例 2-1 使用 LOAD 每日摄入数据。

- 在业务日结束后，运营系统中包含 50 万到两亿条记录的多个平面文件如期到来。
- 客户脚本使用 DB2 LOAD 实用程序将每个文件装载到事实表中。这是在每夜批处理窗口中当表离线的时候完成的。
- 下一个业务日一开始，查询这个表的用户就可以看到前一天的数据。

例 2-2 使用 insert 的准实时摄取。

- 每过 30 分钟，一个包含 1 万到 10 万条记录的文件到来。
- 当该文件到来时，用户编写的程序将这些记录添加到 staging 表中，然后使用 insert 语句将这些记录添加到事实表中。

例 2-3 MQT 刷新。

在有 MQT 的时候，它们被看作将新数据添加到数据仓库这个过程中的一部分。比较特别的是，这种情况下可以使用 MQT 刷新策略。通常，ETL 过程是手动地指定何时更新 MQT，而不是让更新自动发生。

- 对于例 2-1，很可能是在每夜更新完所有任务之后立即更新。
- 对于例 2-2，MQT 通常是每天更新一次。因此，即使底层的数据是周期性更新的，访问 MQT 的查询全天返回的也都是相同的结果。

2) DB2 分区特性发挥的作用

DB2 分区特性对于转入会有帮助，但是有时候也会带来新情况，客户在转入过程中要适当考虑这一点。

通过 DPF 可以更快速地添加数据，因为每个数据库分区可以并行工作。另一方面，DPF 又需要考虑将行发送到适当的数据分区。

与不使用 MDC 相比，使用 MDC 可以改善转入过程。其优点包括：

- 更少的物理 I/O：MDC 表的 RID 索引更少，因此在转入期间更新索引时需要的物理 I/O 更少。
- 更快的插入：MDC 表减少了页面争用和锁，因此有助于使用多个并行的流来执行插入。

- 并发的业务查询能拥有更好的性能：MDC 表减少了页面争用的锁，这一点同时也有利于提高并发业务查询的性能。

另一方面，如果使用 MDC，建议预先根据 MDC 维对数据进行排序。

在某些情况下，TP 有助于转入操作。TP 允许将行添加到一个分区，然后再在准备好的时候将那个分区附加(attach)到表上。然而，在这里的例子中，这个选项并不适用。还记得吗？我们的示例表(Transactions 历史表)对于每个月都有一个单独的分区，而我们是每天一次或多次添加数据。在这种情况下，在一个月开始之前，即这个月还没有开始每天添加数据之前，需要为这个表附加(attach)一个空白的分区。

最后，MQT 还增加了转入过程中要考虑的因素。特别是，需要决定何时更新 MQT。

8. 关键考虑：删除数据

1) 对这一考虑的描述

当数据在数据仓库中存放了一段时间后，对于业务用户来说就不再有用了，因此需要删除它们，为新数据腾出空间。这个过程称作转出、清洗(purging)或归档。下面列出了可能遇到的各种不同情况。

通常，转出涉及以下业务规则，并关系到如何使用 DB2 分区特性的问题：

- 删除过了一定时间的行：这是最简单也是最常见的业务需求。在传统的历史表中，一般的期限为 36 个月。对于最近历史表，一般期限为 60 到 180 天。
- 根据业务规则删除到了一定时间的行：在这种情况下，有些行虽然到了一般的清除时间，但是仍然需要保留。例如，为了为争议或调查提供证据，可能需要保留某个历史事务(例如银行需要保存数据以备人民银行检查)。
- 使数据仍然可以被访问，但是释放存储：这种情况有时候称作归档，而不是转出。在这种情况下，行必须对用户查询可见，但是需要将这些很少被访问的行转移到更便宜的、性能更低的存储上。

通常，MQT 也需要放进来考虑。MQT 也需要更新以删除相应的汇总数据。例如，如果从事实表中删除了 2012 年 3 月的数据，那么也需要删除 MQT 中关于那个月的汇总数据。

2) DB2 分区特性发挥的作用

为了支持转出，针对不同的业务需求，DB2 分区有一些值得注意的特性。

先说 DPF，这个特性对转出作用不大。使用 DPF 与不使用 DPF 相比，转出(roll-out)操作相差不大。

MDC 和 TP 都能为转出操作带来好处。在任何 DB2 版本中，一个特性可能比另一个特性能提供更好的转出性能，但是随着时间的推移，这两个特性应该大致相当。在比较这些特性时，更重要的是看每个特性在某些转出情况下特有的优势。

对于基本的、常见的转出情况(也就是说，只是删除到了一定时间的行)，TP 显然是首选。如果使用 TP，那么转出可以通过简单的 DETACH 操作来完成。而且，TP 是唯一适合以下情况的特性：

- 将转出的数据移动到另外一个位置(也就是说移动到表或数据库中)，而不仅仅是删除它们。
- 根据业务逻辑把较老的、很少被访问的行转移到更便宜的存储中，但是用户查询仍然可以看到它们。

MDC 是唯一适合以下情况的特性，与基本转出相比，下面这些情况要求更高，但是不大常见：

- 客户希望在并发查询活动期间执行转出，但是他们不能接受在 DETACH 操作期间 TP 暂时持有的超级排它锁的影响。
- 客户不希望修改他们的应用程序或脚本。也就是说，他们不想用 TP 的 DETACH 语句代替他们的 DELETE 语句。
- 客户想删除除时间维(表在这一列上进行数据分区)以外的其他维上的大量数据。
- 客户想根据业务规则删除到了一定时间的行。在这种情况下，他们可以发出一条 SELECT 语句来识别符合条件的行，然后再 DELETE 那些行。

对于 MQT，当从 MQT 中删除相应的汇总数据时，建议在 MQT 上使用表分区，并与基本表定义一样的数据分区。

9. 删除数据的其他选项

虽然转出是删除过时数据的常见方式，但是应该注意到，客户有时候也使用其他方式来删除数据，这些方式不需要借助分区特性。这些方式有：

- 刷新表：在某些数据仓库中，整个表每年才删除一次，然后装载用于替代的表，这个新表包含了除不再需要的数据以外的所有数据。
- 清洗：在某些最近历史表中，每过一些天就删除整个表，并重新创建表，数据被放在有更长历史的另一个表中。然后，重新创建的空表又可以摄取新的数据，直到清洗日的到来。

上面我们介绍了 DB2 的表设计特性：表分区、MDC、DPF 和 MQT。协同使用这些特性，可帮助我们解决在查询性能、插入新数据和删除过期数据方面关心的问题。表 2-13 总结了这些高级设计特性是如何解决各种客户需求的。

表 2-13　DB2 特性如何解决客户需求

客 户 需 求	优　　点
查询性能	每个特性都以自己的方式为提高查询性能做出贡献，使用更多的特性将导致更好的性能
转入	在大多数客户场景中，MDC 可以带来最大的好处，TP 可以在某些不常见的情况下带来好处
转出	对于简单、常见的转出场景，TP 可以带来最大的好处，MDC 则适合处理 TP 不适合的其他转出情况

注意：

一般情况下，选择数据库分区的可能性比较小，首先是因为在 DB2 V9 之前由于没有表分区，数据库分区是我们唯一和无奈的选择。而在 DB2 V9 有了表分区以后，相对来说选择数据库分区就不是必需的了。其次，如果部署数据库分区，就必须购买高昂的 DPF 授权，并且在部署起来相当复杂。当然，如果我们的数据量非常大，并且单台机器的硬件资源已经达到了极限，这种情况下可以考虑使用数据库分区。

2.6　行压缩

在信息化时代，很多企业现在生成的数据比以往任何时候都要多。为了遵从法律法规，很多企业(例如银行、移动公司、证券公司等)必须将数据保留更长的时间。这样做的结果是数据库以不可思议的速度增长。

数据的暴增给企业的存储、保护、发布以及从海量数据中获取价值带来了极大的压力。表压缩的主要目的是节省存储空间，但是也可以大大节省磁盘 I/O 并提高缓冲池命中率，因而可以提高性能。这需要一些额外的成本——数据压缩和解压缩需要占用额外的 CPU 资源。表压缩的存储节省和性能影响与数据、数据库本身的设计、数据库调优程度以及应用程序负载有关。下面我们解释这项 DB2 高级设计技术的工作方式，并展示如何启用表压缩。

2.6.1　概念

行压缩是 DB2 V9 里新增加的一项功能，在 DB2 V9.7 里又有了进一步的增强，在 DB2 V10 之后，DB2 又增加了自适应压缩功能，进一步提高了压缩效率，可以说是 DB2 的一大竞争利器。行压缩通过将各行之间重复的值模式替换为更短的符号字符串来压缩数据行，在 DB2 提供的所有压缩工具里，行压缩是压缩率最高的，在我所经历的项目里，通过行压缩平均都可以节省 60%~70%的空间。节省空间的另外一个好处就是带来了更加高效的 I/O 性能，在存在 I/O 瓶颈的系统中，通过压缩可以降低数据页个数，因此就降低了磁盘扫描的次数。因此在大型分析型系统中，强烈建议启用行压缩这一特性。

行压缩使用基于静态字典的压缩算法来逐行压缩数据。字典用来将表行中重复的字节模式映射至更小的符号；这时，这些符号替换表行中更长的字节模式。压缩字典与表数据行一起存储在表的数据对象部分。数据字典本身非常小，大约 100KB，因此数据字典的采样效率决定了压缩率的高低。压缩前和压缩后的数据变化如图 2-6 所示。

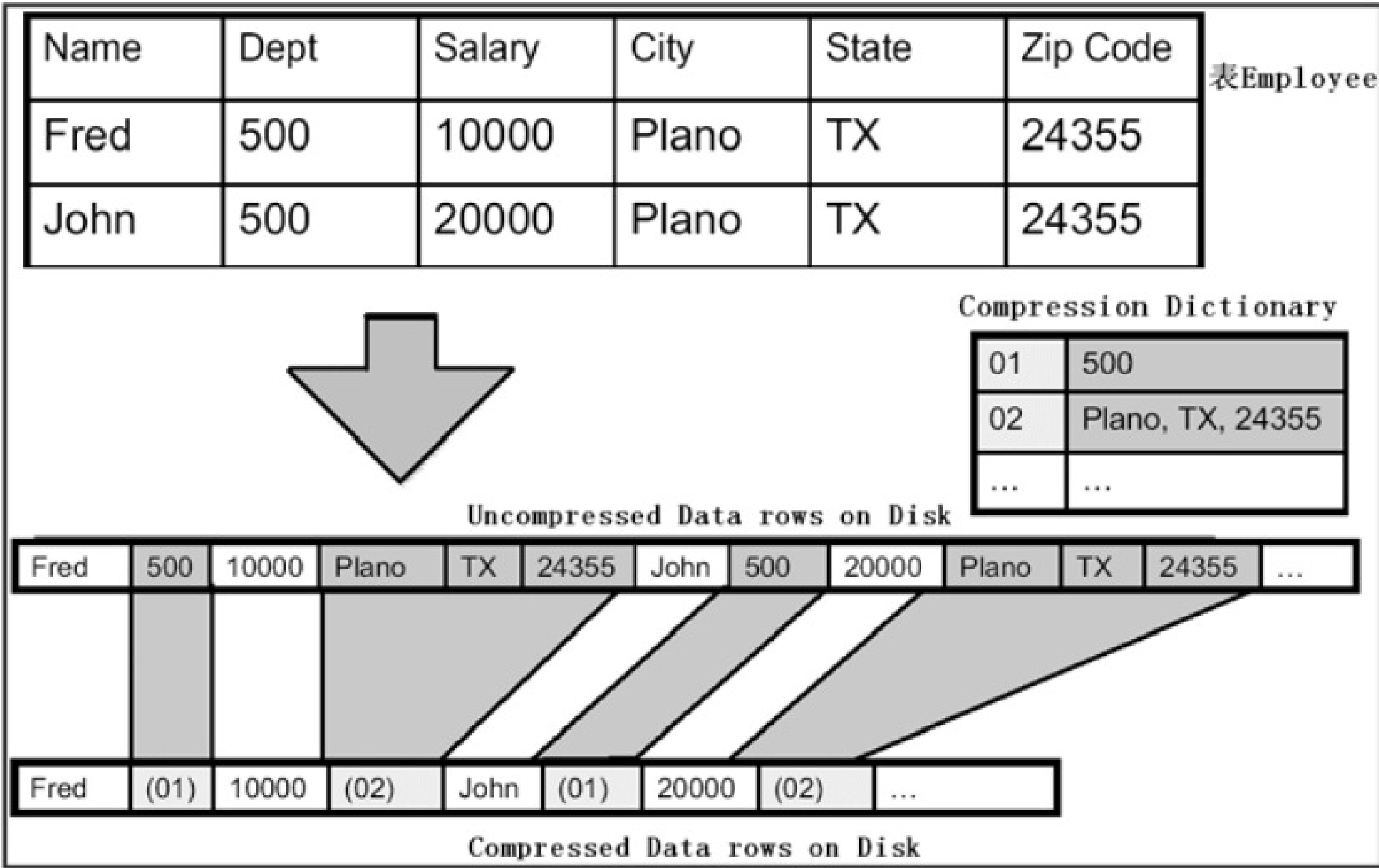


图 2-6　压缩前和压缩后的数据比较

要对表进行表压缩，必须满足以下两个先决条件：

- 必须在表级别启用压缩
- 必须存在表的压缩字典

在 DB2 V9.7 以及之后的版本里，除了可以对数据进行压缩外，下面的类型也可以进行压缩：

- 索引
- 临时表
- XML

自适应压缩实际使用两种压缩方法。第一种方法使用经典行压缩中使用的同一表级别压缩字典，以根据表中数据抽样的重复情况来整体压缩数据。第二种方法使用基于页级别字典的压缩算法以根据每个数据页中的数据重复情况来压缩数据。这些字典将重复字节模式映射至小得多的符号；然后这些符号会替换表中的较长字节模式。表级别压缩字典存储在为其创建该字典的表对象中，用于压缩整个表的数据。页级别压缩字典与数据页中的数据存储在一起，用于仅压缩该页中的数据。

注意：

只能使用表级别压缩字典来指定使用经典行压缩对表进行压缩。但是，不能仅使用页级别压缩字典来指定对表进行压缩。自适应压缩同时使用表级别压缩字典和页级别压缩字典。

2.6.2　启用或禁用表的压缩功能

我们可以通过在 CREATE TABLE 或 ALTER TABLE 语句里指定 COMPRESS YES ADAPTIVE/STATIC 参数来启用行压缩，STATIC 指定只使用表级别的压缩模式，ADAPTIVE 指定使用表级别和页级别的压缩模式。启用压缩功能之后，用于对表添加数据的操作(例如 INSERT、LOAD INSERT 或 IMPORT INSERT 操作)可以使用压缩功能。在 DB2 中，对于在压缩表上创建的索引，如果没有特殊指定，那么也将默认检查压缩索引。

```
CREATE TABLE TEST.SALES
( ID                   INTEGER NOT NULL,
  SALES PERSON         VARCHAR(50),
  REGION              VARCHAR(50),
  SALES DATE           DATE)
COMPRESS YES ADAPTIVE;

create index test.idx1 on test.sales(id);
```

创建完压缩表后，我们可以通过查询 syscat.tables 和 syscat.indexes 的系统视图来检查一下是否启动了行压缩：

```
    $ db2 "select ROWCOMPMODE,compression from syscat.tables where
tabname='SALES' and tabschema='TEST'"

    ROWCOMPMODE COMPRESSION
    ----------- -----------
    A          R

      1 record(s) selected.

    $db2 "select COMPRESSION from syscat.indexes where indname='IDX1'"

    COMPRESSION
    -----------
    Y

      1 record(s) selected.
```

syscat.tables 的 ROWCOMPMODE 字段有下面 5 个取值：

- A = 动态压缩

- S = 静态压缩
- 空 = 行压缩未启用

syscat.tables 的 COMPRESSION 字段有下面 5 个取值：

- B = 值压缩和行压缩同时启用
- N = 没有启用任何压缩
- R = 行压缩启用
- V = 值压缩启用
- Blank = 不适用

syscat.indexes 的 COMPRESSION 字段只有 Y 和 N 两个值，用于标识是否启用了行压缩。

我们可以通过在 ALTER TABLE 语句里指定 COMPRESS NO 参数来禁用行压缩。禁用后将不会对以后添加的行进行压缩。要对整个表进行解压缩，必须使用 REORG TABLE 命令执行离线表重组。

```
    $db2 "alter table test.sales compress no"
    DB20000I  The SQL command completed successfully.

    $ db2 "select ROWCOMPMODE,compression from syscat.tables where
tabname='SALES' and tabschema='TEST'"

    ROWCOMPMODE COMPRESSION
    ----------- -----------
              N

      1 record(s) selected.

    $db2 "select COMPRESSION from syscat.indexes where indname='IDX1'"

    COMPRESSION
    -----------
    Y

      1 record(s) selected.
```

此时索引仍然处于压缩状态，因此必须单独调用 alter index … compress no 命令来解压缩索引：

```
    $db2 "alter index test.idx1 compress no"
    DB20000I  The SQL command completed successfully.
```

```
$db2 "select COMPRESSION from syscat.indexes where indname='IDX1'"

COMPRESSION
-----------
N

  1 record(s) selected.
```

对表使用 alter table … compress yes 也有类似的情况，需要单独对索引调用 alter index … compress yes 命令来启用压缩索引：

```
$db2 "alter table test.sales compress yes"
DB20000I  The SQL command completed successfully.
$db2 "alter index test.idx1 compress no"
DB20000I  The SQL command completed successfully.
```

2.6.3　创建数据字典

数据库管理器将对每个支持行压缩的表创建压缩字典。此字典用来将表行中重复的字节模式映射至更小的符号；这时，这些符号替换表行中更长的字节模式。

行压缩逻辑将对表进行扫描以查找重复的数据。将检查所有行(而不仅仅是检查这些行的某些字段或某些部分)，以了解是否存在重复的条目或模式。收集重复的条目之后，DB2 数据库将构建压缩字典，并为这些条目指定简短的数字键。包含文本数据的表可以具有重复的字符串、具有重复字符的数据以及前导或尾部空格。与包含数字数据的表相比，数据库管理器可以更有效地压缩这些表。

从 DB2 V9.5 开始，数据的压缩字典可以自动创建也可以手动创建。满足下面 3 个条件的话就可以自动创建压缩：

- 已将表的 COMPRESS 属性设置为 YES。可以在创建表时使用 CREATE TABLE 语句的 COMPRESS YES 选项将此属性设置为 YES；也可以在 ALTER TABLE 语句中使用此选项将现有表改变为使用压缩功能。
- 该表尚不存在压缩字典。
- 表已经达到一定大小，有足够的数据用于构造重复数据的字典。

自动创建压缩字典的过程如图 2-7 所示。

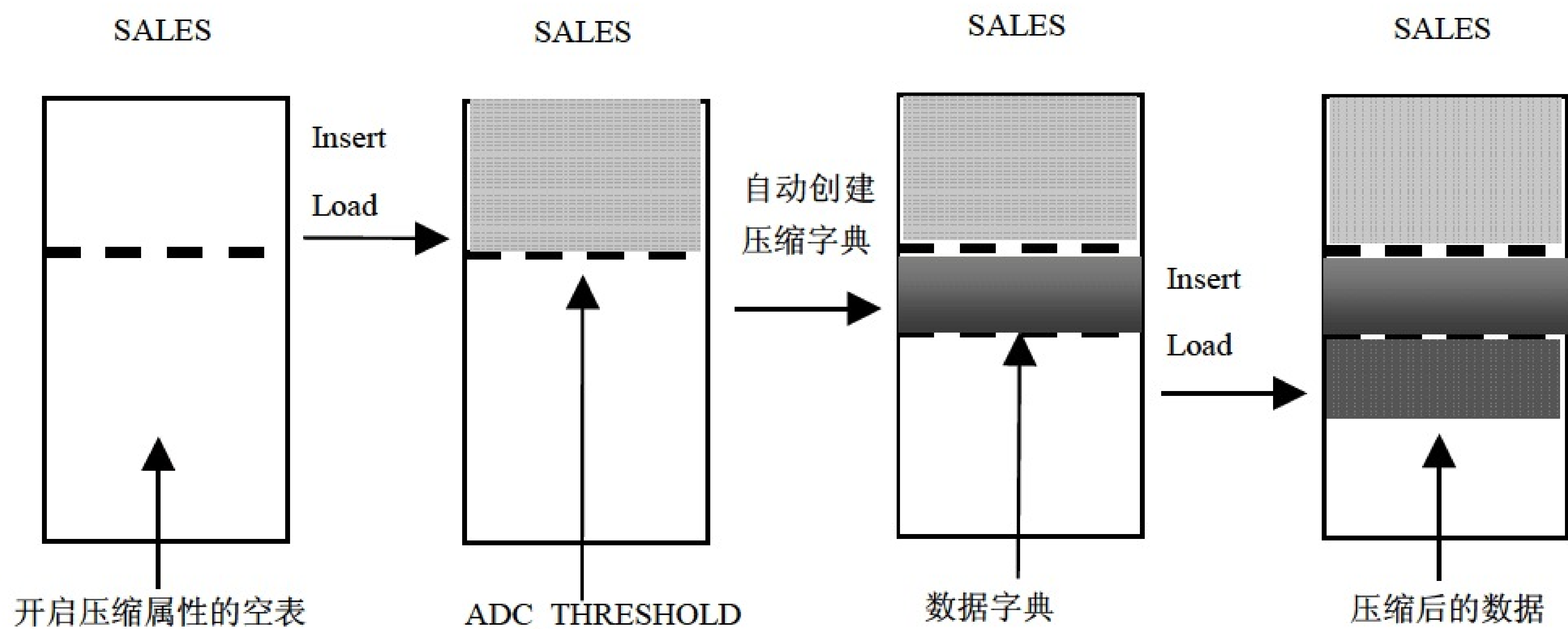


图 2-7　自动创建压缩字典

(1) 将 SALES 表的压缩属性打开。

(2) 加载一定的数据达到 ADC_THRESHHOLD，默认大约是 1MB~2MB。可以通过 db2set 的 DB2_ROWCOMP_ADCTHRESH 参数进行修改。

(3) 自动触发 DB2 对已存在的数据取样，创建压缩字典。

(4) 后面新加载的数据都经过压缩字典进行压缩后存储。

注意：

只有创建字典后的数据才是压缩数据，如果要对创建字典前的数据进行压缩，请手工创建字典。

尽管压缩字典可以自动创建，但是如果不存在压缩字典，也可以使用 REORG 命令强制建立压缩字典或者使用带有 RESETDICTIONARY 选项的 REORG 命令重建压缩字典。对于不带 RESETDICTIONARY 的 REORG 命令，如果表已经启用压缩并且压缩字典不存在，那么 DB2 会在数据量达到 1MB~2MB 时创建数据字典，对于小于 1MB 的表，如果要强制创建数据字典，需要显式指定 RESETDICTIONARY 命令。

KEEPDICTIONARY 和 RESETDICTIONARY 与压缩字典的关系如表 2-14 和表 2-15 所示：

表 2-14　KEEPDICTIONARY 与压缩字典的关系

Compress	Dictionary Exists	result; outcome
Y	Y	保留压缩字典；所有行被压缩
Y	N	创建压缩字典；所有行被压缩
N	Y	保留压缩字典；所有行被解压缩
N	N	没有效果

表 2-15　RESETDICTIONARY 与压缩字典的关系

Compress	Dictionary Exists	result; outcome
Y	Y	重建压缩字典；所有行被压缩
Y	N	创建新的压缩字典；所有行被压缩
N	Y	删除压缩字典；所有行被解压缩
N	N	无效果

2.6.4　评估压缩空间

在真正对表进行压缩重组之前，为了评估执行重组后的压缩率，我们可以使用带有 ESTIMATE 参数的 ADMIN_GET_TAB_COMPRESS_INFO_V97 表函数或 INSPECT 工具来查看通过行压缩为表节省的存储器容量估计值。该评估过程如下：

(1) 对要评估的表执行 runstats 统计分析：

```
$db2 "runstats on table test.sales on all columns and detailed indexes all"
DB20000I  The RUNSTATS command completed successfully.
```

(2) 执行带有 ESTIMATE 参数的 ADMIN_GET_TAB_COMPRESS_INFO_V97 命令：

```
db2 "SELECT COMPRESS ATTR, ROWS SAMPLED, PAGES SAVED PERCENT FROM TABLE
(SYSPROC.ADMIN GET TAB COMPRESS INFO ('TEST', 'SALES', 'ESTIMATE'))"

COMPRESS ATTR ROWS SAMPLED PAGES SAVED PERCENT
------------- ------------ --------------------
N                       208                  60
```

COMPRESS_ATTR 为 N 代表没有启用行压缩。

ROWS_SAMPLED 为 208 代表对 208 行进行了采样。

PAGES_SAVED_PERCENT 为 60 代表采用行压缩可以节省 60%的存储器容量。

2.6.5　检查压缩状态

DB2 提供了 syscat.tables 系统视图和带有 REPORT 选项的 ADMIN_GET_TAB_COMPRESS_INFO 命令，用于返回已压缩表的压缩状态信息。举例如下：

(1) 对要评估的压缩表执行 RUNSTATS 统计分析：

```
$db2 "runstats on table test.sales on all columns and detailed indexes all"
DB20000I  The RUNSTATS command completed successfully.
```

(2) 使用 syscat.tables 系统视图返回压缩信息：

```
    db2 "select
compression,pctpagessaved,AVGROWCOMPRESSIONRATIO,PCTROWSCOMPRESSED from
syscat.tables where tabname='SALES'"

    COMPRESSION PCTPAGESSAVED AVGROWCOMPRESSIONRATIO   PCTROWSCOMPRESSED
    ----------- ------------- ------------------------------------------------
    R                  60          +2.75000E+000          +1.00000E+002

      1 record(s) selected.
```

COMPRESSION 为 R 代表启用行压缩。

PCTPAGESSAVED 为 60 代表采用行压缩可以节省 60%的存储器容量。

AVGROWCOMPRESSIONRATIO 代表压缩前的容量与压缩后的容量比。

PCTROWSCOMPRESSED 代表多少行被压缩。

(3) 执行带有 REPORT 参数的 ADMIN_GET_TAB_COMPRESS_INFO_V97 命令：

```
    db2 "SELECT COMPRESS ATTR, ROWS SAMPLED, PAGES SAVED PERCENT FROM TABLE
(SYSPROC.ADMIN GET TAB COMPRESS INFO ('TEST', 'SALES', 'REPORT'))"

    COMPRESS ATTR ROWS SAMPLED PAGES SAVED PERCENT
    ------------- ------------ -------------------
    Y                    208                60
```

2.6.6　行压缩应用案例

下面我们举两个应用表压缩的实际案例。

案例一：比较常规表空间和大型表空间、常规 RID 和大型 RID 对压缩的影响

下面的例子使用 SAMPLE 数据库的 ORDERS 表(大小总共为 1GB)来展示在使用压缩之后 RID 数量受到的影响。这里使用一个平均行大小较小的表。表空间使用 16 KB 的页大小来存储表 ORDERS。使用这个页大小时，可以很好解释大型 RID 和常规 RID 的区别。

主要步骤如下：

(1) 在 DB2 V8 中，在常规表空间中创建表 ORDERS，检查页数、平均行大小和每页行数。

(2) 将实例和数据库迁移至 DB2 V9。

(3) 在 DB2 V9 中创建新的大型表空间。

(4) 在新的大型表空间中创建和 ORDERS 表类似的 ORDERS2 表。

(5) 压缩表 ORDERS 和 ORDERS2，并分别在这两个表上运行 REORG。

(6) 现在可以比较这两个表的页数、平均行大小和每页行数。可以看到常规表空间与大型表空间之间存在不同之处。

例 2-4 显示了表 ORDERS 的结构。

例 2-4　表 ORDERS 的结构。

```
db2inst1@bmcc:~/tmp/test> db2 describe table orders
Column                       Type      Type
name                         schema    name                Length   Scale Nulls
------------------------------ --------- ------------------ --------- -----
O ORDERKEY                   SYSIBM    INTEGER              4         0     No
O CUSTKEY                    SYSIBM    INTEGER              4         0     No
O ORDERSTATUS                SYSIBM    CHARACTER            1         0     No
O TOTALPRICE                 SYSIBM    DECIMAL             15         2     No
O ORDERDATE                  SYSIBM    DATE                 4         0     No
O ORDERPRIORITY              SYSIBM    CHARACTER           15         0     No
O CLERK                      SYSIBM    CHARACTER           15         0     No
O SHIPPRIORITY               SYSIBM    INTEGER              4         0     No
O COMMENT                    SYSIBM    VARCHAR             20         0     No
  9 record(s) selected.
db2inst1@bmcc:~/tmp/test>
```

平均行大小(查询中的 AVG_ROW_SIZE 列)为 79 字节，见例 2-5。列 ROWS_PER_PAGE 是通过将行数(CARD)除以页数(NPAGES)得到的。

例 2-5　常规表空间中的表 ORDERS。

```
db2inst1@bmcc:~/tmp/test> db2 -tvf avg row size.sql
SELECT SUBSTR(a.tabname,1,10) AS table, b.npages ,
CASE WHEN (b.NPAGES  > 0) THEN (b.CARD / b.NPAGES) ELSE -1 END AS ROWS
PER PAGE,
SUM(AVGCOLLEN) AVG ROW SIZE
FROM SYSCAT.COLUMNS a, SYSCAT.TABLES b, SYSCAT.TABLESPACES c
WHERE a.tabschema = b.tabschema AND a.tabname = b.tabname
AND b.tbspaceid = c.tbspaceid
AND a.tabname = 'ORDERS'
GROUP BY a.tabschema, a.tabname, pagesize, card, npages
TABLE     NPAGES              ROWS PER PAGE       AVG ROW SIZE
---------- -------------------- -------------------- ------------
ORDERS     8198                 182                        79
  1 record(s) selected.
db2inst1@bmcc:~/tmp/test>
```

在 DB2 V8 的常规表空间中，在无压缩模式下，每页存储 182 行。总共有 8198 页，这接近于每页 255 行的限制，所以 16 KB 的页大小是个很好的选择。在将实例和数据库迁移至 DB2 V9 之后，必须更新统计信息。新的页大小为 16 KB 的大型表空间 tbs_16K 将被创建，在新的大型表空间中将创建与表 ORDERS 类似的新表 ORDERS2。SAMPLE 数据库中没有使用索引或约束，因此表的创建非常简单。数据将由游标装载，如例 2-6 所示。

例 2-6 创建表 ORDERS2。

```
db2inst1@bmcc:~/tmp/test> ./cr_tbspace_tbs_16K.sh
 Database Connection Information
 Database server        = DB2/AIX 9.1.0
 SQL authorization ID   = DB2INST1
 Local database alias   = SAMPLE
 DROP TABLESPACE tbs_16K
 DB21034E  The command was processed as an SQL statement because it was not a
 valid Command Line Processor command.  During SQL processing it returned:
 SQL0204N  "TBS_16K" is an undefined name.  SQLSTATE=42704
 CREATE LARGE TABLESPACE tbs_16K PAGESIZE 16 K MANAGED BY DATABASE
 USING ( FILE '/db2/db2inst1/TPCH/tbs_16K.001' 20000 ) BUFFERPOOL bp16k
DB20000I  The SQL command completed successfully.
CREATE TABLE orders2 LIKE orders IN tbs_16K
DB20000I  The SQL command completed successfully.
DB20000I  The SQL command completed successfully.
DECLARE c1 CURSOR FOR SELECT * FROM orders
DB20000I  The SQL command completed successfully.
LOAD FROM c1 OF CURSOR INSERT INTO orders2
SQL3501W The table space(s) in which the table resides will not be placed in
backup pending state since forward recovery is disabled for the database.
SQL1193I  The utility is beginning to load data from the SQL statement "
SELECT * FROM orders".
SQL3500W  The utility is beginning the "LOAD" phase at time "2007-02-09
23.03.50.673543".
SQL3519W  Begin Load Consistency Point. Input record count = "0".
SQL3520W  Load Consistency Point was successful.
SQL3110N The utility has completed processing.  "1500000" rows were read from
the input file.
SQL3519W  Begin Load Consistency Point. Input record count = "1500000".
SQL3520W  Load Consistency Point was successful.
```

```
SQL3515W  The utility has finished the "LOAD" phase at time "2008-02-19
23.04.17.263410".
Number of rows read         = 1500000
Number of rows skipped      = 0
Number of rows loaded       = 1500000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 1500000
db2inst1@bmcc:~/tmp/test>
```

在 DB2 V9 的大型表空间中，在无压缩模式下，每页也是存储 182 行。平均行大小(79)、页数(8198)与常规表空间中的一样，如例 2-7 所示。

例 2-7　大型表空间中 ORDERS2 表的数字。

```
db2inst1@bmcc:~/tmp/test> db2 -tvf avg row size.orders2.sql
SELECT SUBSTR(a.tabname,1,10) AS table, PAGESIZE, b.CARD, b.npages ,
CASE WHEN (b.NPAGES  > 0) THEN (b.CARD / b.NPAGES) ELSE -1 END AS
ROWS PER PAGE,
SUM(AVGCOLLEN) AVG ROW SIZE FROM SYSCAT.COLUMNS a, SYSCAT.TABLES b,
SYSCAT.TABLESPACES c
WHERE a.tabschema = b.tabschema
AND a.tabname = b.tabname
AND b.tbspaceid = c.tbspaceid
AND a.tabname = 'ORDERS2'
GROUP BY a.tabschema, a.tabname, pagesize, card, npages
TABLE     NPAGES                ROWS PER PAGE       AVG ROW SIZE
---------- -------------------- -------------------- ------------
ORDERS2    8198                  182                  79
  1 record(s) selected.
db2inst1@bmcc:~/tmp/test>
```

DB2 V9 中的系统编目表包含名为 AVGROWSIZE 的新列，这个列显示表中压缩行和无压缩行的平均大小(单位为字节)。例 2-8 分别针对表 ORDERS 和 ORDERS2 检索这个列。这两个表的平均行大小均为 89 字节。这个值与计算出的 79 字节略微有些差别，计算值是通过计算所有列的 AVGCOLLEN 得到的。

例 2-8　检索无压缩模式下的 AVGROWSIZE。

```
db2inst1@bmcc:~/tmp/test> db2 "select AVGROWSIZE
from syscat.tables where tabname = 'ORDERS2' "
AVGROWSIZE
```

```
----------
       89
  1 record(s) selected.
db2inst1@bmcc:~/tmp/test>
db2inst1@bmcc:~/tmp/test> db2 "select AVGROWSIZE
from syscat.tables where tabname = 'ORDERS' "
AVGROWSIZE
----------
       89
  1 record(s) selected.
db2inst1@bmcc:~/tmp/test>
```

现在对这两个表进行压缩，如例 2-9 所示。

注意：
表的重组是必需的，ALTER TABLE 语句只是更新编目的内容。

例 2-9 压缩表。

```
db2inst1@bmcc:~/tmp/test> db2 "alter table db2inst1.orders compress yes"
DB20000I  The SQL command completed successfully.
db2inst1@bmcc:~/tmp/test>
db2inst1@bmcc:~/tmp/test> db2 "alter table db2inst1.orders2 compress yes"
DB20000I  The SQL command completed successfully.
db2inst1@bmcc:~/tmp/test>
db2inst1@bmcc:~/tmp/test> time db2 -v "reorg table db2inst1.ORDERS
             resetdictionary"
reorg table  db2inst1.ORDERS resetdictionary
DB20000I  The REORG command completed successfully.
real  0m58.335s
user  0m0.018s
sys 0m0.029s
db2inst1@bmcc:~/tmp/test>
db2inst1@bmcc:~/tmp/test> time db2 -v "reorg table  db2inst1.ORDERS2
             resetdictionary"
reorg table  db2inst1.ORDERS2 resetdictionary
DB20000I  The REORG command completed successfully.
real  0m59.505s
user  0m0.020s
```

```
sys 0m0.028s
db2inst1@bmcc:~/tmp/test> db2 "reorgchk update statistics
on table db2inst1.orders " > reorgchk.orders.compr.out
db2inst1@bmcc:~/tmp/test> db2 "reorgchk update statistics
on table db2inst1.orders2 " > reorgchk.orders2.compr.out
db2inst1@bmcc:~/tmp/test>
```

在例 2-10 中可以看到，常规表空间中的页数现在是 253，仍然受每页 255 行的限制。但是，在大型表空间中，对于表 ORDERS2，每页可以有 427 行。因此，表 ORDERS2 使用的页数少于表 ORDERS 使用的页数。当压缩常规表空间中的表时，效果就显现出来了。为了避免常规表空间中每页 255 行的这个限制，必须使用大型表空间，然后就可以在压缩之后在每页存储更多的行。

例 2-10　压缩后的数字。

```
db2inst1@bmcc:~/tmp/test> db2 -tvf avg_row_size.sql
SELECT SUBSTR(a.tabname,1,10) AS table, PAGESIZE, b.CARD, b.npages ,
CASE WHEN (b.NPAGES  > 0) THEN (b.CARD / b.NPAGES)
ELSE -1 END AS ROWS_PER_PAGE,
SUM(AVGCOLLEN) AVG ROW SIZE
FROM SYSCAT.COLUMNS a, SYSCAT.TABLES b, SYSCAT.TABLESPACES c
WHERE a.tabschema = b.tabschema
AND a.tabname = b.tabname
AND b.tbspaceid = c.tbspaceid
AND a.tabname = 'ORDERS'
GROUP BY a.tabschema, a.tabname, pagesize, card, npages
TABLE     NPAGES              ROWS PER PAGE       AVG ROW SIZE
---------- -------------------- -------------------- ------------
ORDERS                   5907                 253         79
  1 record(s) selected.
db2inst1@bmcc:~/tmp/test> db2 -tvf ars.orders2.sql
SELECT SUBSTR(a.tabname,1,10) AS table, PAGESIZE, b.CARD, b.npages ,
CASE WHEN (b.NPAGES  > 0) THEN (b.CARD / b.NPAGES)
ELSE -1 END AS ROWS PER PAGE,
SUM(AVGCOLLEN) AVG_ROW_SIZE
FROM SYSCAT.COLUMNS a, SYSCAT.TABLES b, SYSCAT.TABLESPACES c
WHERE a.tabschema = b.tabschema
AND a.tabname = b.tabname
AND b.tbspaceid = c.tbspaceid
AND a.tabname = 'ORDERS2'
```

```
GROUP BY a.tabschema, a.tabname, pagesize, card, npages
TABLE      NPAGES                ROWS_PER_PAGE        AVG_ROW_SIZE
---------- -------------------- -------------------- ------------
ORDERS2                    3512                  427          79
  1 record(s) selected.
db2inst1@bmcc:~/tmp/test>
```

如例 2-11 所示，平均行大小(AVGROWSIZE)仍然相同。现在平均行大小是 38 字节，在不压缩的情况下这个值为 89 字节(见例 2-11)。

例 2-11 检索 AVGROWSIZE。

```
db2inst1@bmcc:~/tmp/test>
db2inst1@bmcc:~/tmp/test> db2 "select AVGROWSIZE
 from syscat.tables where tabname = 'ORDERS' "
AVGROWSIZE
----------
        38
  1 record(s) selected.
db2inst1@bmcc:~/tmp/test> db2 "select AVGROWSIZE
 from syscat.tables where tabname = 'ORDERS2' "
AVGROWSIZE
----------
        38
  1 record(s) selected.
db2inst1@bmcc:~/tmp/test>
```

随着压缩特性的引入，编目中增加了一些新的列。特别是，列 PCTPAGESSAVED 显示了压缩后节省的页数的比率。在这个案例中，在大型表空间中节省了 57%的页，因为使用的页数从 8198 减至 3512。常规表空间需要更多的页。压缩后，常规表空间的页数可以从 8198 减至 5907，如例 2-12 所示。使用大型表空间相对常规表空间可以节省 40.5%的页(从 5 907 减至 3 512)。

例 2-12 两个表各自节省的页。

```
    db2inst1@bmcc:~/tmp/test> db2 "select SUBSTR(tabname,1,20) AS table, npages,
from syscat.tables where tabschema = 'DB2INST1' and tabname LIKE  'ORDERS%'"
    TABLE                NPAGES               PCTPAGESSAVED
    -------------------- -------------------- ------------------------
    ORDERS                              5907                    27
    ORDERS2                             3512                    57
      2 record(s) selected.
    db2inst1@bmcc:~/tmp/test>
```

表 2-16 对上述结果做了总结。使用大型表空间可以在最大程度上节省空间。压缩之后，只需要 3512 页，而常规表空间仍然需要 5907 页。常规表空间与大型表空间之间的差距是 5907 页-3512 页，换言之，前者浪费了 2395 页。

表 2-16　页大小为 16KB 时每页行数的比较

表	表　空　间	模　　式	CARD 行数	NPAGES	AVG ROWSIZE	AVG ROWS PER PAGE
ORDERS	REGULAR	No compression	1 500 000	8198	89	182
ORDERS	REGULAR	Compressed	1 500 000	5907	38	253
ORDERS2	LARGE	No compression	1 500 000	8198	89	182
ORDERS2	LARGE	Compressed	1 500 000	3512	28	427

表 2-17 比较了压缩后各表节省的空间。在表 ORDERS 中，压缩后节省了 28%的页；而在表 ORDERS2 中，压缩后节省了 57%的页。

表 2-17　比较节省的空间

表	表　空　间	PCTPAGESSAVED	压缩前 NPAGES	压缩后 NPAGES	SAVINGS
ORDERS	REGULAR	27	8198	5907	28
ORDERS2	LARGE	570	8198	3512	57

案例总结：

经过上面的案例，我们得出的结论是，在 DB2 V9 以后应尽量使用大型表空间，因为大型表空间支持大型 RID，可以在数据页中存放更多的行数(常规是 255 行)，同时在容量上可以最大到 16TB(常规最大是 256GB)。同时，大型表空间在压缩比上也远远优于常规表空间。

案例二：测试表进行数据压缩的装载和查询效率

例 2-13　未进行数据压缩表：

```
CREATE TABLE "ODSUSR"."PB_CSTLOG_1"(
    "LOGNO" VARCHAR(20) NOT NULL ,
    "TRANDATE" DATE  NOT NULL ,
    "TRANTIME" TIME  NOT NULL ,
```

```
    "CSTNO" CHARACTER(10) NOT NULL ,
    "ACCNO" VARCHAR(30) NOT NULL ,
    "TRANCODE" VARCHAR(20) NOT NULL ,
    "RESULT" VARCHAR(8),
    "INFO" VARCHAR(500),
    "IP" VARCHAR(20),
    "MAC" VARCHAR(100),
    "ISIMPORTENT" VARCHAR(2)
) in TBS_ODSDATA;
```

例 2-14 数据压缩表：

```
CREATE TABLE "ODSUSR"."PB_CSTLOG_2"(
    "LOGNO" VARCHAR(20) NOT NULL ,
    "TRANDATE" DATE  NOT NULL ,
    "TRANTIME" TIME  NOT NULL ,
    "CSTNO" CHARACTER(10) NOT NULL ,
    "ACCNO" VARCHAR(30) NOT NULL ,
    "TRANCODE" VARCHAR(20) NOT NULL ,
    "RESULT" VARCHAR(8),
    "INFO" VARCHAR(500),
    "IP" VARCHAR(20),
    "MAC" VARCHAR(100),
    "ISIMPORTENT" VARCHAR(2)
) in TBS_ODSDATA COMPRESS YES
```

例 2-15 建立数据压缩字典：

```
REORG table PB_CSTLOG_2 resetdictionary;
```

详细测试记录如表 2-18 所示。

表 2-18 详细测试记录

测试项目	记录数	字段数	行平均长度	压缩比	装载时间	查询时间
未数据压缩	7648864	11	177	0	36.172	54.188
数据压缩	7648864	11	65	63	49.235	54.359

我们可以看到，经过压缩而节省的页数所占的百分比为 63%，通过压缩而节省的字节数所占的百分比也为 63%，但是数据压缩后加载的时间比压缩前长了一些，而查询的时间

相差不大。在查询期间还发现压缩的表数据在查询期间 CPU 负载比未压缩前明显要高很多(在笔者的测试例子中，大概高 13%左右)。

经过上面两个案例，我们已经知道表压缩的优点和限制，为了充分利用 DB2 V9 中行压缩的优点，应遵循以下最佳实践：

- 在迁移至 DB2 V9 之后更新统计信息(通过运行 RUNSTATS 或 REORGCHK UPDATE STATISTICS 命令)。
- 使用 DB2 INSPECT 命令估计表能节省的空间大小，以决定哪些表值得压缩。
- 行压缩之后，为了使压缩生效，应对表进行重组。
- 从 DB2 V8 迁移至 DB2 V9 时，要计划从常规表空间迁移至大型表空间。
- 当迁移至大型表空间时，要同时考虑安排数据移动和数据重组。
- 大的归档历史表比较适合压缩。

2.6.7　索引压缩及应用案例

可以压缩索引(包括已声明的临时表或者已创建的临时表的索引)，以便降低存储器开销。对于大型 OLTP 和数据仓库环境而言，此功能特别有用。

在默认情况下，索引压缩功能对于已压缩的表处于启用状态，对于未压缩的表处于禁用状态。可以使用 CREATE INDEX 语句的 COMPRESS YES 选项来更改此默认行为。处理现有索引时，使用 ALTER INDEX 语句来启用或禁用索引压缩；然后必须执行索引重组以重建索引。

限制： 下列类型的索引不支持索引压缩功能：

- MDC 块索引
- XML 路径索引
- 不能对索引规范进行压缩
- 无法使用 ALTER INDEX 命令对临时表的索引更改压缩属性

启用索引压缩功能后，将根据数据库管理器选择的压缩算法对索引页在磁盘上和内存中的格式进行修改，以便最大程度地减少存储空间耗用量。根据所创建索引的类型以及索引所包含数据的不同，实现的压缩程度也会有所变化。例如，通过存储重复键的记录标识(RID)的缩写格式，数据库管理器可以对包含大量重复键的索引进行压缩。在索引键前缀的公共程度很高的索引中，数据库管理器可以根据索引键前缀的相似性来进行压缩。

存在一些与压缩相关联的限制和权衡。如果索引未共享公共索引列值或者不完整的公共前缀，那么索引压缩在减少耗用存储量方面的优势可以忽略不计。并且，尽管时间戳记列的唯一索引可能由于同一个叶子页包含年、月、日、小时、分钟甚至秒的公共值而具有非常高的压缩能力，但检查是否存在公共前缀的开销也会导致性能下降。

如果相信压缩不会对您所处的特定情况带来好处，那么可以在不进行压缩的情况下重新创建索引，也可以更改索引并接着执行索引重组以禁用索引压缩功能。

在考虑使用索引压缩功能时，应该记住下列事项：

- 如果使用 CREATE TABLE 或 ALTER TABLE 命令的 COMPRESS YES 选项启用行压缩功能，那么在默认情况下，将对此后为该表创建的所有支持压缩的索引启用压缩功能，除非通过 CREATE INDEX 或 ALTER INDEX 命令显式禁用该功能。同样，如果通过 CREATE TABLE 或 ALTER TABLE 命令禁用行压缩功能，那么将对此后为该表创建的所有索引禁用索引压缩功能，除非通过 CREATE INDEX 或 ALTER INDEX 命令显式启用该功能。
- 如果使用 ALTER INDEX 命令来启用索引压缩功能，那么直到执行索引重组之后才会进行压缩。同样，如果禁用压缩功能，那么在执行索引重组之前，该索引将保持处于已压缩状态。
- 在数据库迁移期间，不会对任何迁移后的索引启用压缩功能。如果要启用压缩功能，那么必须使用 ALTER INDEX 命令，然后执行索引重组。
- 索引压缩或解压所需进行的处理可能会导致 CPU 使用率略微提高。如果这种情况不可接受，那么可以对新索引或现有索引禁用索引压缩功能。

例 2-16 检查索引是否处于压缩状态。

下面这两条语句将创建支持行压缩功能的新表 T1 并对 T1 创建索引 I1：

```
CREATE TABLE T1 (C1 INT, C2 INT, C3 INT) COMPRESS YES
CREATE INDEX I1 ON T1(C1)
```

默认情况下，T1 的索引处于压缩状态。可以使用目录表或管理表函数来检查索引 T1 的压缩属性，此属性将指示是否已启用压缩功能：

```
SELECT COMPRESSION FROM SYSCAT.INDEXES WHERE TABNAME='T1'
COMPRESSION
-----------
Y
   1 record(s) selected.
```

例 2-17 确定已压缩的索引是否要求进行重组。

要确定已压缩的索引是否要求进行重组，请使用 REORGCHK 命令。下面显示如何对名为 Temp 的表运行此命令：

```
REORGCHK ON TABLE SCHEMA1.Temp
```

```
    Doing RUNSTATS ....
    Table statistics:
    F1: 100 * OVERFLOW / CARD < 5
    F2: 100 * (Effective Space Utilization of Data Pages) > 70
    F3: 100 * (Required Pages / Total Pages) > 80

    SCHEMA.NAME      CARD    OV    NP    FP ACTBLK    TSIZE  F1  F2  F3 REORG
    ----------------------------------------------------------------------
    Table: SCHEMA1.Temp
                      879     0    14    14     -     51861   0 100 100
    ----------------------------------------------------------------------


    Index statistics:

    F4: CLUSTERRATIO or normalized CLUSTERFACTOR > 80
    F5: 100 * (Space used on leaf pages / Space available on non-empty leaf pages)
> MIN(50, (100 - PCTFREE))
    F6: (100 - PCTFREE) * (Amount of space available in an index with one less
level / Amount of space required for all keys) < 100
    F7: 100 * (Number of pseudo-deleted RIDs / Total number of RIDs) < 20
    F8: 100 * (Number of pseudo-empty leaf pages / Total number of leaf pages)
< 20

    SCHEMA.NAME                 INDCARD  LEAF ELEAF LVLS  NDEL    KEYS
LEAF_RECSIZE NLEAF_RECSIZE LEAF_PAGE_OVERHEAD NLEAF_PAGE_OVERHEAD
PCT_PAGES_SAVED  F4  F5  F6  F7  F8 REORG
    ----------------------------------------------------------------------
    Table: SCHEMA1.Temp
    Index: SCHEMA1.Temp
                      879   15     0    2     0     682          20           20
    596               596              28  56  31   -   0   0
---------------------------------------------------------------------------
```

例 2-18　确定索引压缩功能有可能节省的空间量。

要获取有关计算索引压缩功能有可能节省的空间量，需要通过表函数 ADMIN_GET_INDEX_COMPRESS_INFO 来计算评估，语句如下：

```
    db2 "SELECT  substr(TABNAME,1,18),substr(INDNAME,1,18),iid,
compress_attr, index_compressed, pct_pages_saved, num_leaf_pages_saved  FROM
TABLE(admin_get_index_compress_info('', 'INST97', 'TEMP', -2, -2)) AS t"
```

详细用法请参考 DB2 信息中心。

2.7　本章小结

本章向大家介绍了数据仓库系统里常用的表的高级技术，比如表分区、MDC、MQT和行压缩的基础知识，介绍了它们的基本概念、用法和常见问题。很多情况下需要结合使用才能发挥最大的效用，希望读者可以好好领会它们的精髓，融会贯通，知其然，更要知其所以然，将其用到真正适合它们的场景中！

第 3 章

数据库安全

数据库安全在当今社会极其重要。对于公司、企业来说，数据库中的数据往往是最重要的资产。在数据库中，这些数据作为商业信息或知识，一旦遭受安全威胁，就将带来难以想象的严重后果。通常，如果敏感的个人数据(例如手机号码、信用卡号和银行账号等)从不安全的系统中被窃取，那么身份盗用、金融诈骗、未经授权地使用信息等恶果也就接踵而来。因此，系统管理员必须持续监控他们的系统，确保系统中采取了适当的安全预防措施。

在系统架构的不同级别上，可以采用不同的技术来进行安全控制。例如，可以通过安装防火墙来防止外部网络对服务器的未经授权的访问。可以使用一些安全网络协议技术，例如 IPSec，保证网络上计算机间通信信道的安全。又如，可以实行严格的密码策略，要求用户选择强密码并经常更换密码。应用系统的安全需要操作系统、网络、应用、中间件和数据库等层面上相应的安全性防范措施协同工作，这样才能构建牢固的安全体系。本章主要讲解和 DB2 数据库有关的安全技术。

本章主要讲解如下内容：

- DB2 安全机制概述
- 认证
- 权限
- 特权
- 某银行安全规划案例

- 执行安全审计
- 基于标签的访问控制
- 安全经验总结

3.1 DB2 安全机制概述

DB2 中有 3 种主要的安全机制，可以帮助 DBA 实现数据库安全计划：身份认证(authentication)、权限(authorization)和特权(privilege)。

1. 身份认证(authentication)

身份认证是用户在尝试访问 DB2 实例或数据库时遇到的第一道安全闸门。身份认证就是使用安全机制验证所提供用户 ID 和口令的过程。用户和密码身份认证由 DB2 外部的设施管理，比如操作系统、域控制器或 Kerberos 安全系统。这和其他数据库管理系统(DBMS)是不同的，如 Oracle、Informix、Sybase 和 SQL Server，后者既可以在数据库本身定义和验证用户账户，也可在外部设施(如操作系统)中完成。外部安全性服务对希望访问 DB2 服务器的用户进行身份认证，DB2 外部的安全性软件负责处理身份认证。当成功校验了用户 ID 和口令后，内部 DB2 进程将接管控制，并确保用户有权执行所请求的操作。

DB2 数据库是没有用户的，所有用户都是操作系统用户，这和别的数据库不一样。别的数据库有数据库用户和操作系统用户两种类型。而 DB2 中用户使用的是操作系统用户。之所以有这个限制，是由历史原因造成的。因为 DB2 最初是从 DB2 for MVS/ESA 平台转变过来的，在该平台上有这个限制，所以用户创建和认证都要用操作系统来完成。

一旦用户 ID 和口令作为实例附件或数据库连接请求的一部分明确地提供给 DB2，DB2 就会尝试使用外部安全设施验证用户 ID 和口令。如果请求中没有提供用户 ID 和口令，那么 DB2 UDB 隐含使用登录到发出请求的工作站时所用的用户 ID 和口令。

实际的认证位置由 DB2 实例参数 AUTHENTICATION 的值决定。有不同的身份认证方案，包括让用户在 DB2 服务器上认证(使用服务器的安全设施)、在客户机上认证(允许“单点登录”访问)、使用 Kerberos 安全设施认证或者用户定义的通用安全服务(Generic Security Service，GSS)插件认证。其他身份认证选项还包括：当用户名和口令以及数据在客户机和服务器之间的网络上传递时进行加密。为 AUTHENTICATION 参数选择的值依赖于具体环境和本地安全策略。

2. 权限(authorization)

权限涉及将 DB2 角色赋予用户、角色或组。每一种角色具有一定级别的权限，可以对

特定数据库或其中的对象执行某些命令。DB2 中包括以下多种不同角色或权限：系统管理员(SYSADM)、系统控制(SYSCTRL)、系统维护(SYSMAINT)和系统监视(SYSMON)、安全性管理员(SECADM)、数据库管理员(DBADM)、访问控制(ACCESSCTRL)、数据访问(DATAACCESS)、SQL 管理员(SQLADM)、工作负载管理管理员(WLMADM)以及说明(EXPLAIN)权限(提供了数据库内的控制权、LOAD、CONNECT 权限)。

3. 特权(privilege)

特权的粒度比授权要细，可以分配给用户、角色或组。特权定义用户可以创建或删除的对象。它们还定义用户可以用来访问对象(比如表、视图、索引和应用程序包)的命令。另外，基于标签的访问控制(LBAC)，特权允许以更细的粒度控制谁有权访问单独的行或列。

4. 安全层次

下面我们用一个例子来说明 DB2 安全模型的实现机制。比如，图 3-1 中的连接语句供用户 *bob* 使用口令 *bobpsw* 连接到 *finance* 数据库。身份认证过程包括下面 7 个步骤：

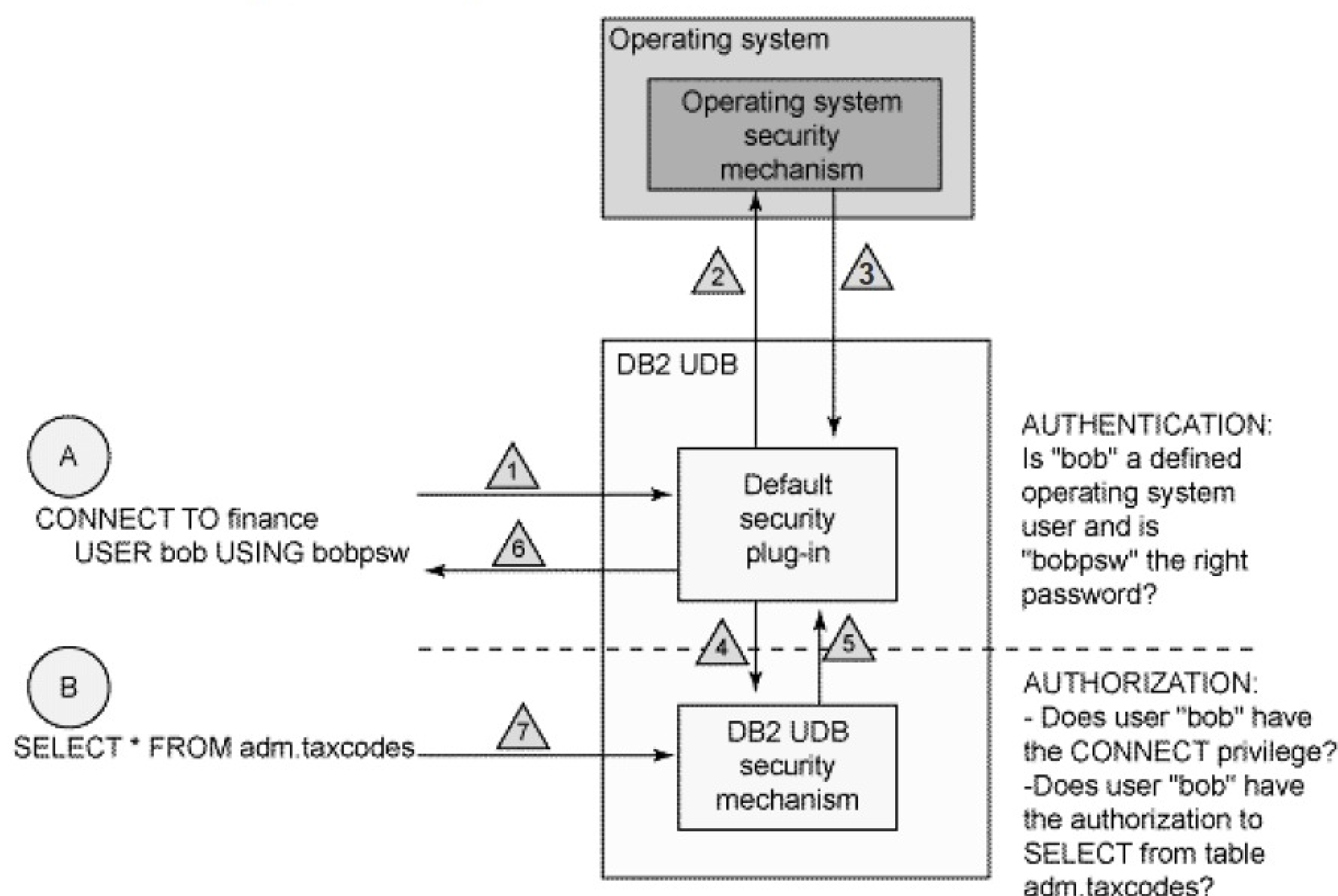


图 3-1　DB2 安全模型的实现机制

(1) CONNECT 语句传递给 DB2 数据库服务器。

(2) 如果没有明确配置安全插件，就使用默认的安全插件。如果包含 *finance* 数据库的实例的 AUTHENTICATION 参数设为 SERVER(默认设置)，那么连接请求中的用户 ID 和口

令由 DB2 数据库服务器上的安全设施验证。默认插件将用户 ID 和口令发送给操作系统进行验证。

(3) 操作系统确认 *bob/bobpsw* 组合是否有效，把该信息返回给安全插件。

(4) 安全插件激活 DB2 安全机制，对用户 *bob* 查询 DB2 系统编目表中和权限相关的表，看看该用户是否被授予了该数据库的 CONNECT 权限。默认情况下，CONNECT 特权被授予 PUBLIC，也就是说任何通过身份认证的用户都能连接数据库。

(5) DB2 安全机制验证用户 *bob*，并且把成功或错误信息返回给安全插件。

(6) 安全插件把成功或失败的消息返回给用户。如果用户没有通过身份认证，DB2 就会拒绝连接请求，并向客户机应用程序返回错误消息，如下所示：

```
SQL30082N  Attempt to establish connection failed with security
reason "24"("USERNAME AND/OR PASSWORD INVALID").  SQLSTATE=08001
```

这时，DB2 服务器上的 DB2 诊断日志(db2diag.log)中也会出现类似于下面这样的记录：

```
2012-09-07-13.52.08.216577+480 I559634A438       LEVEL: Warning
PID     : 49086688            TID  : 15696      PROC : db2sysc 0
INSTANCE: db2inst1             NODE : 000         DB   : TEST
APPHDL  : 0-1469
EDUID   : 15696               EDUNAME: db2agent(TEST)0
FUNCTION: DB2 UDB, bsu security, sqlexLogPluginMessage, probe:20
DATA #1 : String with size, 67 bytes
Password validation for user bob failed with rc = -2146500507
```

如果遇到这样的消息，一定要确认连接到数据库的用户或应用程序是否提供了合法的用户 ID 和口令。该用户 ID 和口令必须存在于执行用户身份认证的设施中(由目标 DB2 UDB 实例的 AUTHENTICATION 参数决定)。

3.2 认证(authentication)

3.2.1 什么时候进行 DB2 身份认证

DB2 身份认证控制数据库安全性策略的以下方面：

- 谁有权访问实例和/或数据库
- 在哪里以及如何检验用户的密码

在发出 *attach* 或 *connect* 命令时，借助于底层操作系统的安全特性实现对 DB2 用户的身份认证。*attach* 命令用来连接 DB2 实例，而 *connect* 命令则用来连接 DB2 实例中的数据库。下面的示例展示了 DB2 对发出这些命令的用户进行身份认证的不同方式，这些示例在

数据库管理程序配置文件中使用默认的身份认证类型 SERVER。最后一个示例说明了如何使用 DB2 修改服务器操作系统上的密码。

用创建 DB2 实例时使用的用户 ID 登录到安装了 DB2 的机器上。发出以下命令：

```
db2 ATTACH TO db2inst1
```

在这里，虽然没有显式地提供用户名和密码，但是隐式地执行了身份认证。使用登录机器的用户 ID，并假设这个 ID 和密码已经经过了操作系统的检验。

```
db2 CONNECT TO test USER db2inst1 USING passwd
Database Connection Information
Database server        = DB2/AIX64 9.7.6
SQL authorization ID   = DB2INST1
Local database alias   = TEST
```

在这里，显式地执行了身份认证。用户 *db2inst1* 和密码 *password* 由操作系统进行检验。用户 *db2inst1* 成功地连接到示例数据库：

```
db2 CONNECT TO test USER db2inst1 USING passwd NEW newpasswd CONFIRM newpasswd
```

与前面的第 2 个示例一样，用户 ID *db2inst1* 和密码 *passwd* 由操作系统进行检验。然后，操作系统将 *db2inst1* 的密码从 *passwd* 改为 *newpasswd*。这时候再使用密码 *passwd* 登录就会失败。

3.2.2　DB2 身份认证类型

1. 客户机与服务器

在考虑整个 DB2 数据库环境的安全性时，理解术语客户机、服务器是相当重要的。数据库环境常常由几台不同的机器组成，必须在所有潜在的数据访问点上保护数据库。在处理 DB2 身份认证时，理解客户机、服务器的概念尤其重要。

图 3-2 说明了基本的客户机-服务器配置。



图 3-2　客户机与服务器

数据库服务器是数据库实际所在的机器(在分区的数据库系统上可能是多台机器)。DB2 数据库客户机是对服务器上的数据库执行查询的机器。这些客户机可以是本地的(驻留

在与数据库服务器相同的物理机器上)，也可以是远程的(驻留在单独的机器上)。

DB2 在实例级使用了身份认证类型参数 AUTHENTICATION，这个参数决定了在什么地方进行身份认证。例如，在客户机-服务器环境中，AUTHENTICATION 参数的设置决定了是在客户机上还是在服务器上检验用户的 ID 和密码。

2. 身份认证类型

DB2 能够根据用户是试图连接数据库，还是执行实例连接和实例级操作，指定不同的身份认证机制。在默认情况下，实例对于所有实例级和连接级请求使用一种身份认证类型。这由数据库管理程序配置参数 AUTHENTICATION 来指定。从 DB2 V9.1 开始，DB2 引入了数据库管理程序配置参数 SRVCON_AUTH，这个参数专门处理对数据库的连接。例如，如果在 DBM CFG 中进行以下设置：

```
db2 GET DBM CFG
Server Connection Authentication        (SRVCON_AUTH)= KERBEROS
Database manager authentication     (AUTHENTICATION)= SERVER_ENCRYPT
```

那么在连接实例时会使用 SERVER_ENCRYPT，但是在连接数据库时会使用 Kerberos 身份认证。如果在服务器上没有正确地初始化 Kerberos，但是提供了有效的用户 ID/口令，那么允许这个用户连接实例，但是不允许连接数据库。身份认证类型确定在何处验证用户 ID/口令对。所支持的主要身份认证类型有：

- SERVER(默认)
- SERVER_ENCRYPT
- Kerberos
- KRB_SERVER_ENCRYPT
- CLIENT

身份认证类型是在服务器和客户机处同时设置的。

服务器

每个实例仅允许一种类型的身份认证，也就是说，设置适用于该实例下定义的所有数据库。在数据库管理器配置文件中使用 AUTHENTICATION 参数指定该设置，以下示例指定在服务器端认证：

```
db2 UPDATE DBM CFG USING AUTHENTICATION server
```

客户机

在客户机上编目的各数据库拥有自己的身份认证类型，用 catalog database 命令中的

AUTHENTICATION 参数指定，以下示例指定在服务器端认证：

```
db2 CATALOG DB test as test2 AT NODE db2inst1 AUTHENTICATION server
```

1) 使用 server 选项进行身份认证

使用 server 选项时，服务器首先检测连接是本地连接还是远程连接。如果是本地连接，那么不需要用户标识和密码；如果是远程连接，用户 ID 和口令将发送到服务器进行校验。考虑以下示例。

(1) 用户使用用户名 *peter* 和口令 *peterpwd* 登录到工作站。

(2) *peter* 随后使用用户 ID *db2user* 和口令 *db2pwd* 连接到 SAMPLE 数据库，这是在远程 DB2 服务器上定义的。

(3) *db2user* 和 *db2pwd* 通过网络发送到服务器。

(4) *db2user* 和 *db2pwd* 在 DB2 服务器上校验。

整个过程如图 3-3 所示。

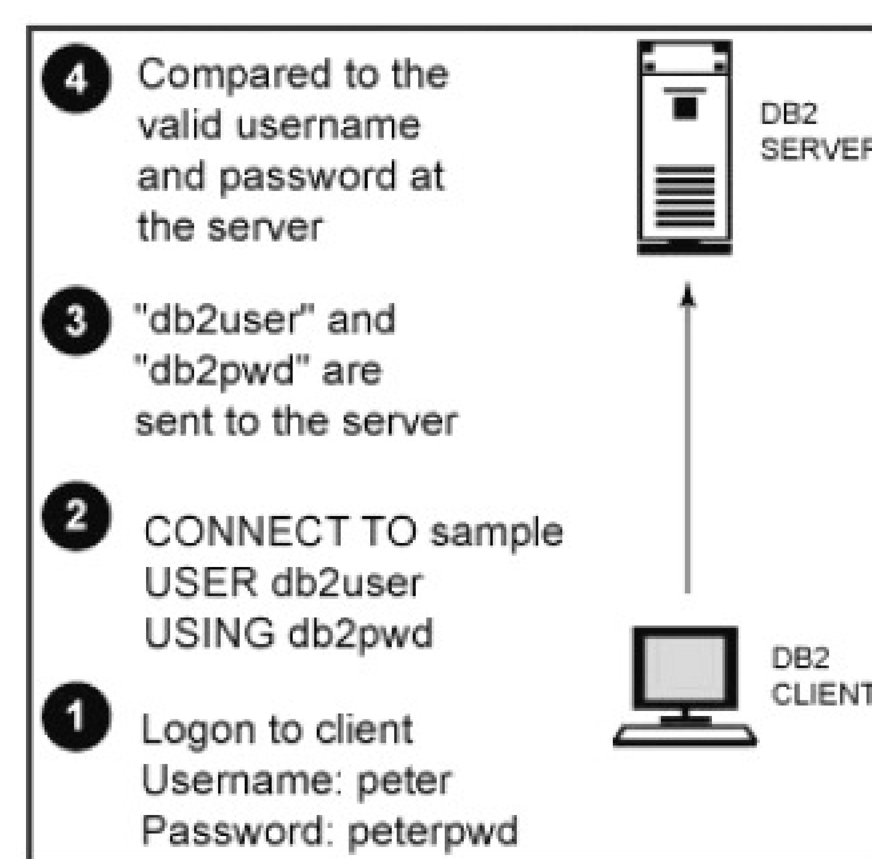


图 3-3　使用 server 选项进行身份认证

如果想避免用户 ID 和口令在网络上被窃听，可使用 SERVER_ENCRYPT 身份认证类型，这样用户 ID 和口令在传送到服务器前会被加密。

2) 使用 Kerberos 进行身份认证

Kerberos 是一种外部安全性协议，使用通用密码术创建共享的加密密钥。Kerberos 安全协议作为第三方身份认证服务执行身份认证，使用传统的密码术创建共享的密钥。共享密钥成为用户的凭证，在请求本地或网络服务时在所有情况下用来检验用户的身份。Kerberos 提供了安全的身份认证机制，这是因为用户 ID 和口令不再需要以明文形式通过网络传输。通过使用 Kerberos 安全协议，可以实现对远程 DB2 数据库服务器的单点登录。

3) 在客户机上进行身份认证

这一选项允许在客户机上进行身份认证。用户成功登录到客户机后，即可轻松连接到数据库，而无需再次提供口令，如图 3-4 所示。

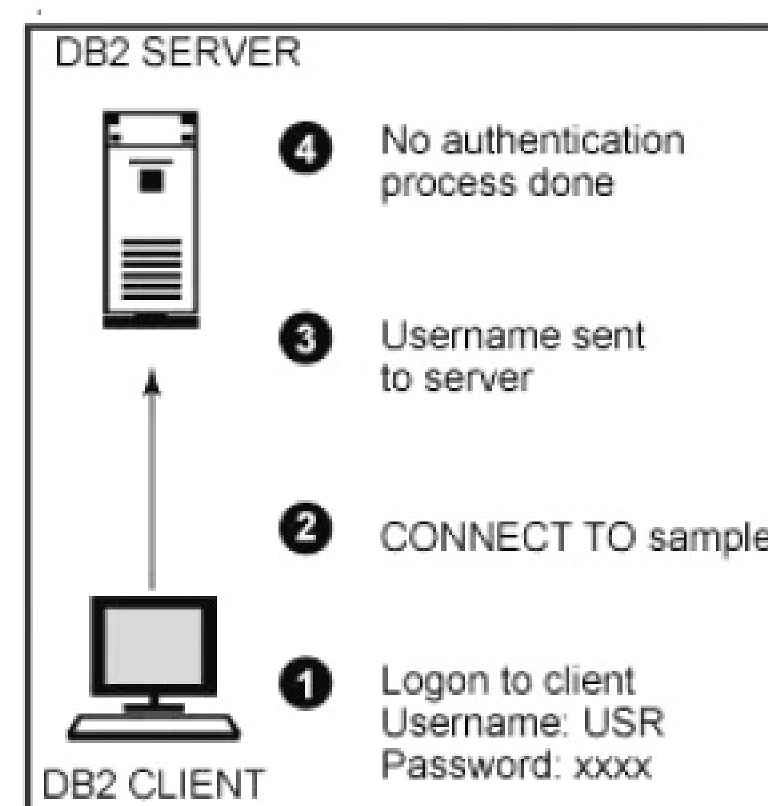


图 3-4　在客户机上进行身份认证

这里有一个重要问题需要考虑：有些客户机系统不具有可靠的安全性设施，例如 Windows 9x 和 Classic Mac OS，它们被称作不受信任的客户机。任何人只要可以访问这些系统，就可以不经过身份认证直接连接到 DB2 服务器。谁知道它们会执行怎样的破

坏性操作(例如删除数据库)？为实现允许受信任的客户机自行执行身份认证，同时强制不受信任的客户机在服务器处进行身份认证的灵活性，DB2 引入了另外两个数据库管理器配置参数：

- TRUST_ALLCLNTS
- TRUST_CLNTAUTH

这两个参数仅在身份认证设置为 CLIENT 时生效。

信任客户机

TRUST_ALLCLNTS 确定信任哪种类型的客户机，该参数有以下 3 种可能值：

- YES——信任所有客户机，这是默认设置。身份认证将在客户机处执行。但有一个例外，我们将在介绍 TRUST_CLNTAUTH 时对此予以详细讨论。
- NO——仅信任具备可靠的安全性设施的客户机(受信任的客户机)。对于不受信任的客户机连接，必须提供用户 ID 和口令，以便在服务器端进行身份认证。
- DRDAONLY——仅信任在 iSeries 或 zSeries 平台上运行的客户机(例如 DRDA 客户机)。其他任何客户机都必须提供用户 ID 和口令。

设想如下场景：DB2 服务器将身份认证设置为 CLIENT，将 TRUST_ALLCLNTS 设置为 YES。作为 *localuser* 登录到一台 Windows XP 计算机，并在未指定用户 ID 和口令的情况下连接到远程数据库。那么 *localuser* 将成为数据库处的连接授权 ID。而如果想使用其他用户 ID 连接到数据库(例如，有执行数据库备份权限的 *poweruser*)，又会怎么样呢？

为允许此类行为，可使用 TRUST_CLNTAUTH 来指定，当在 connect 语句或 *attach* 命令中提供了用户 ID 和密码时将在何处进行身份认证。允许使用的值有两个：

- CLIENT——身份认证在客户机处执行，不需要用户 ID 和口令。
- SERVER——身份认证在服务器处完成，需要提供用户 ID 和口令。

下面的示例说明了如何在服务器和客户机上设置身份认证类型和参数。

在服务器上设置身份认证：

```
db2 UPDATE DBM CFG USING AUTHENTICATION client
db2 UPDATE DBM CFG USING TRUST ALLCLNTS yes
db2 UPDATE DBM CFG USING TRUST_CLNTAUTH server ---重启实例 db2start 以生效
```

在客户机上设置身份认证：

```
db2 CATALOG DB sample AT NODE db2inst1 AUTHENTICATION client
```

在上面的示例中，如果从任何客户机发出如下命令：

```
db2 CONNECT TO sample
```

那么身份认证在客户机上进行。

如果从任何客户机发出如下命令：

```
db2 CONNECT TO sample USER db2inst1 USING passwd
```

那么身份认证在服务器上进行。

4) 其他身份认证设置

如果查看 DB2 版本的实例中的 DBM CFG，就会看到影响 DB2 对用户 ID 进行身份认证的方式的各种设置。正如前面提到的，有针对标准操作系统用户 ID 身份认证的设置(CLIENT、SERVER、SERVER_ENCRYPT、DATA_ENCRYPT、DATA_ENCRYPT_CMP)，还有将身份认证工作交给外部程序的插件(KERBEROS、KRB_SERVER_ENCRYPT、GSSPLUGIN、GSS_SERVER_ENCRYPT)。下面专门介绍其他一些配置变量如何影响对用户的身份认证：

```
Client Userid-Password Plugin        (CLNT PW PLUGIN)=
Group Plugin                        (GROUP PLUGIN)=
GSS Plugin for Local Authorization  (LOCAL GSSPLUGIN)=
Server Plugin Mode                 (SRV PLUGIN MODE)= UNFENCED
Server List of GSS Plugins    (SRVCON GSSPLUGIN LIST)=
Server Userid-Password Plugin      (SRVCON PW PLUGIN)=
Cataloging allowed without authority (CATALOG NOAUTH)= NO
Bypass federated authentication         (FED_NOAUTH)= NO
```

在下面的列表中，不包括已经讨论过的参数。

- CLNT_PW_PLUGIN：这个参数在客户端的 DBM CFG 中指定，指定用来进行客户机和本地身份认证的客户机插件的名称。
- GROUP_PLUGIN：默认值是空(NULL)。如果设置为用户定义的插件，就会调用这个插件进行所有组枚举，而不依赖于操作系统的组查找。这与后面讨论的授权部分相关。
- LOCAL_GSSPLUGIN：这个参数指定默认的 GSS-API 插件库的名称，当数据库管理程序配置的身份认证参数值设置为 GSSPLUGIN 或 GSS_SERVER_ENCRYPT 时，这个库用来进行实例级的本地授权。
- SRV_PLUGIN_MODE(YES/NO)：这个参数的默认设置是 NO。当改为 YES 时，以 FENCED 模式启动 GSS-API 插件，其工作方式类似于 FENCED 存储过程。FENCED 插件的崩溃不会导致 DB2 实例崩溃。在插件还处于开发阶段时，建议以 FENCED 模式运行它们。这样一来，插件中的逻辑问题和内存泄漏就不会导致实例崩溃。在确定插件是可靠的之后，应该以 UNFENCED 模式运行以提高性能。

- SRVCON_GSSPLUGIN_LIST：插件列表，当使用 Kerberos、KRB_SERVER_ENCRYPT、GSSPLUGIN 或 GSS_SERVER_ENCRYPT 时，服务器上的数据库管理程序在身份认证期间将使用这些插件。列表中的每个插件应该用逗号(，)分隔，它们之间没有空格。插件按照优先次序列出，首先使用列表中的第一个插件对发送的用户 ID/口令进行身份认证。只有当列出的所有插件都返回了错误时，DB2 才会向用户返回身份认证错误。
- SRVCON_PW_PLUGIN：在指定 CLIENT、SERVER 或 SERVER_ENCRYPT 身份认证时，这个参数允许用户修改DB2用来检验用户ID和密码的默认身份认证方法。在默认情况下，值是 NULL，因此使用默认的 DB2 方法。
- CATALOG_NOAUTH(YES/NO)：默认值是 NO。将这个参数修改为 YES，可允许不属于 SYSADM、SYSCTRL 或 SYSMAINT 组成员的用户修改机器上的 Database、Node、Admin 和 DCS 编目。登录的用户使用不可信客户机，或者登录所用的用户 ID 不允许连接数据库或实例，但是用户又必须在客户机上进行编目，只有在这种情况下这个参数才是有用的。
- FED_NOAUTH：如果 FED_NOAUTH 设置为 YES，身份认证设置为 SERVER 或 SERVER_ENCRYPT，联邦设置为 YES，那么在实例上避免进行身份认证，假设身份认证在数据源上进行。当 FED_NOAUTH 设置为 YES 时系统会发出警告。在客户机和 DB2 服务器上都不进行身份认证。知道 SYSADM 身份认证名称的任何用户都拥有联邦服务器的 SYSADM 权限。

3.3 权限(authorization)

3.3.1 权限层次

DB2 定义了权限层次结构，用于将一组预先确定的管理权限授予用户账号组。这些管理权限包括能够对数据库进行备份、更改配置参数、查看表数据等等。权限级别控制执行数据库管理器维护操作和管理数据库对象的能力。

DB2 授权控制数据库安全策略的以下方面：

- 用户被授予的权限级别
- 允许用户运行的命令
- 允许用户读取和/或修改的数据
- 允许用户创建、修改和/或删除的数据库对象

按照权限的作用范围来区分，DB2 中共包括两类权限：实例级权限和数据库级权限。

在实例级别定义的权限会应用于这个实例中的所有数据库上；而在数据库级别定义的权限仅应用到特定的数据库，对同一实例中的其他数据库不会产生影响，不同级别的权限关系请参见图 3-5。

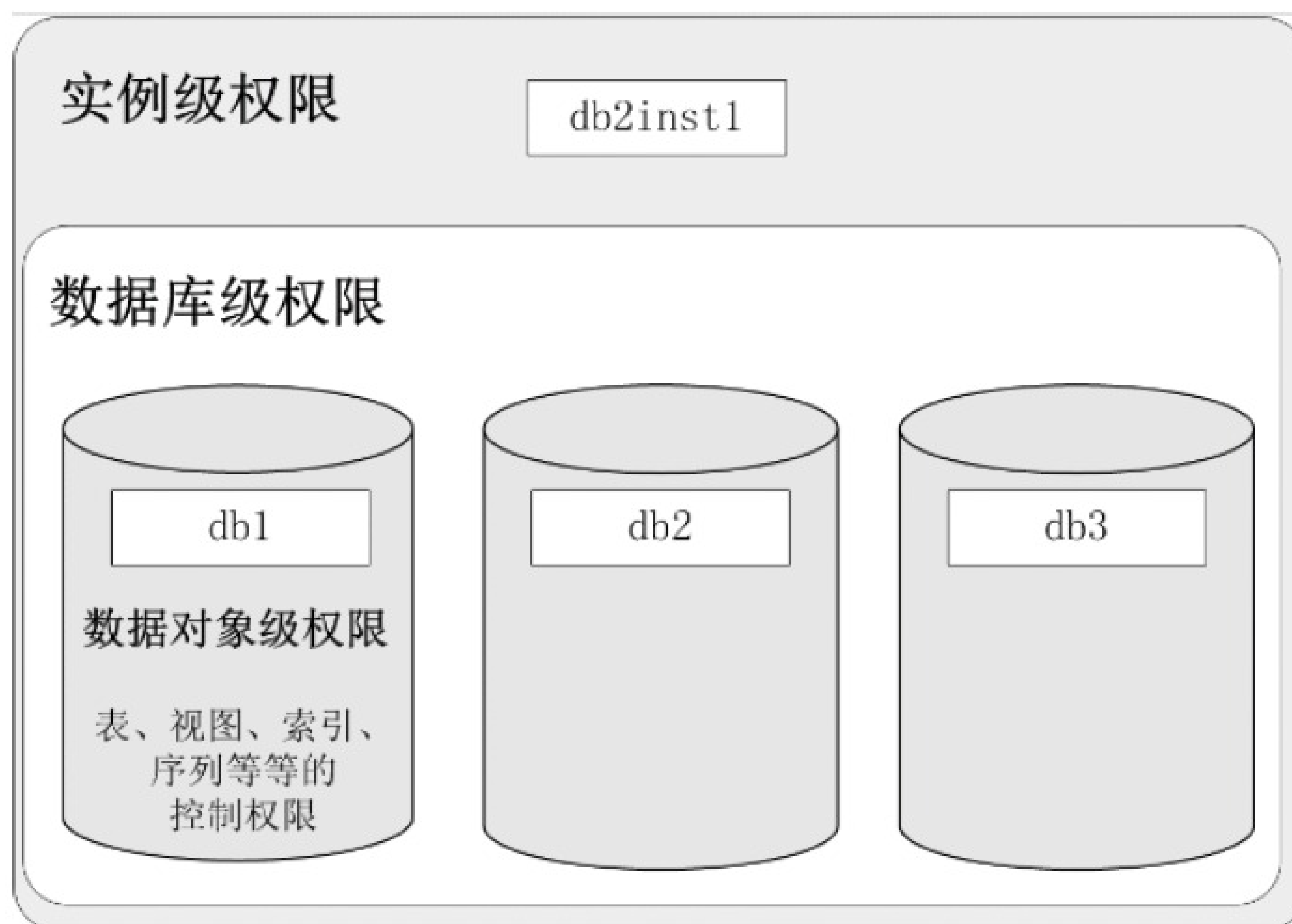


图 3-5　不同级别权限的作用范围

3.3.2　实例级权限

实例级权限包括 SYSADM、SYSCTRL、SYSMAINT、SYSMON，这意味着上述 4 种权限的作用范围包括实例级命令以及针对这个实例中所有数据库的命令。这些权限只能分配给组，可以通过 DBM CFG 文件分配这些权限。

注意：

本章中任何提到组成员关系的地方都假设在操作系统级别已经定义了这些用户和组名。

权限级别按照图 3-6 所示的层次结构进行组织。这个层次结构的顶部是 SYSADM 权限级别，在 DB2 V9.7 以前的版本里，SYSADM 是最高权限，可以执行所有可用的 DB2 操作。但在 DB2 V9.7 中，SYSADM 权限的部分功能已经被分离给 SECADM。SYSCTRL 和 SYSMAINT 权限级别提供了 SYSADM 权限的子集，可以管理系统，但是不允许访问表中的任何数据。SYSMON 权限提供了使用数据库系统监视器的能力。DBADM 在 DB2 V9.7 中也被分为两部分：DATAACCESS，允许对数据库中的所有数据进行访问以及修改；ACCESSCTRL，管理部分数据库级权限，以及数据库里所有数据对象权限的授予和撤回，但自身不能访问编目表和视图以外的数据。LOAD 权限允许用户运行 LOAD 实用程序，这

是 DB2 UDB 的高速批量数据装载器。SECADM 用于管理数据库内的安全性，例如 DBADM 权限、特权的授予与撤回、审计等。



图 3-6　权限的层次结构

表 3-1 总结了每个权限的级别及用途。

表 3-1　权限级别及用途总结

权限级别	说明和用途
SYSADM	基本拥有 DB2 所有的权限，并可以指定拥有 SYSADM、SYSCTRL、SYSMAINT 和 SYSMON 系统权限的组。在 DB2 V9.7 中，DBADM 已经从 SYSADM 中分离，创建数据库会默认给当前用户此库的 DBADM 权限
SYSCTRL	最高的系统控制权限级别。提供对数据库管理器实例及其数据库执行维护和管理操作的能力，例如创建、删除数据库和表空间等 不允许直接访问数据库中的数据。具有连接数据库的隐式特权，并可以执行具有 SYSMAINT 和 SYSMON 权限的用户能够执行的功能。该权限供管理包含敏感数据的数据库管理器实例的用户使用
SYSMAINT	次高的系统控制权限级别，提供对数据库管理器实例及其数据库执行维护和管理操作的能力，不允许直接访问数据库中的数据。具有连接数据库的隐式特权，并可以执行具有 SYSMON 权限的用户能够执行的功能。该权限供维护包含敏感数据的数据库管理器实例中数据库的用户使用
SYSMON	提供获得数据库管理器实例及其数据库的快照的能力。如果数据库管理器实例中的数据库包含敏感数据，而管理用户只需要通过快照监控数据来进行问题判断，这时候就可以给该用户授予 SYSMON 权限。该权限不允许改变系统资源的使用

表 3-2 对比了每个权限级别允许的常见管理操作。

表 3-2　每个权限级别允许的常见管理操作的比较

功　　能	SYSADM	SYSCTRL	SYSMAINT	SYSMON
UPGRADE DATABASE	YES	NO	NO	NO
GRANT/REVOKE DBADM	NO	NO	NO	NO
UPDATE DBM CFG	YES	NO	NO	NO
CHANGE SYSCTRL/SYSMAINT AUTHORITY	YES	NO	NO	NO
UPDATE DB/NODE/DCS DIRECTORIES	YES	YES	NO	NO
FORCE USERS OFF DATABASE	YES	YES	NO	NO
CREATE/DROP DATABASE	YES	YES	NO	NO

(续表)

功　　能	SYSADM	SYSCTRL	SYSMAINT	SYSMON
CREATE/DROP/ALTER TABLE SPACE	YES	YES	NO	NO
RESTORE TO NEW DATABASE	YES	YES	NO	NO
UPDATE DB CFG	YES	YES	YES	NO
BACKUP DATABASE OR TABLE SPACE	YES	YES	YES	NO
RESTORE TO EXISTING DATABASE	YES	YES	YES	NO
PERFORM ROLLFORWARD RECOVERY	YES	YES	YES	NO
START/STOP DATABASE INSTANCE	YES	YES	YES	NO
RESTORE TABLE SPACE	YES	YES	YES	NO
RUN TRACE	YES	YES	YES	NO
OBTAIN MONITOR SNAPSHOTS	YES YES	YES YES	YES YES	YES YES
CREATE/ACTIVATE/DROP EVENT MONITOR	NO	NO	NO	NO
QUERY TABLE SPACE STATE	YES	YES	YES	YES
PRUNE LOG HISTORY FILES	YES	YES	YES	NO
QUIESCE INSTANCES	YES	YES	NO	NO
QUIESCE DATABASES	YES	NO	NO	NO
QUIESCE TABLE SPACE	YES	YES	YES	NO
REORG TABLE	YES	YES	YES	NO
RUN RUNSTATS UTILITY	YES	YES	YES	NO
LOAD TABLE	YES	NO	NO	NO
READ TABLE DATA	YES	NO	NO	NO

获得实例级权限的办法是：将在外部安全设施中定义的用户组赋给相关的实例级权限参数(SYSADM_GROUP、SYSCTRL_GROUP、SYSMAINT_GROUP、SYSMON_GROUP)。例如，如果希望用户账号 xinzhuang 具有 SYSMAINT 权限，那么可以将 xinzhuang 放进 MAINT 组，然后将实例参数 SYSMAINT_GROUP 更新为 MAINT。这样，MAINT 组中的

任何用户都将具有 SYSMAINT 权限。要从 xinzhuang 那里撤销 SYSMAINT 特权，只需要从 MAINT 组中删除它，或者将 SYSMAINT_GROUP 参数的值改为另一个不包含它的组。在后一种情况下，如果 MAINT 组的其他成员不是新组的成员，那么它们的 SYSMAINT 权限也会被撤销。

实例级权限的参数除了使用命令行修改外，也可通过控制中心进行修改。

SYSADM 权限

在 DB2 V9.7 以前的版本里，DB2 中的 SYSADM 权限基本上就像是 UNIX 上的 root 权限或 Windows 上的 Administrator 权限。对 DB2 实例拥有 SYSADM 权限的用户能够对这个实例、这个实例中的任何数据库以及这些数据库中的任何对象发出任何 DB2 命令，他们还能够访问数据库中的数据，以及对其他用户授予或撤销特权或权限。但是为了保证数据的安全，需要把访问数据的权限从管理实例的权限中分离，于是 IBM 在 DB2 V9.7 中把 SYSADM 权限中对数据库里对象的访问以及数据库对象上的管理权限分离给了数据库管理员(DBADM)和安全管理员(SECADM)。另外，只允许 SYSADM 用户更新 DBM CFG 文件。SYSADM 权限由 DBM CFG 文件中的 SYSADM_GROUP 参数控制。在 Windows 上，在创建实例时，这个参数设置为 Administrator(如果安全时启用操作系统安全性选项，该组为 db2admins)。但是，如果发出命令 db2 get dbm cfg，那么它看起来是空的。在 UNIX 上，它设置为创建这个实例的用户的主组。

因为只允许 SYSADM 用户更新 DBM CFG 文件，所以只有他们能够向其他组授予任何 SYS*权限。以下示例演示了如何向 *db2grp1* 组授予 SYSADM 权限：

```
db2 UPDATE DBM CFG USING SYSADM_GROUP db2grp1
```

请记住，这一修改直到实例停止并重新启动之后才会生效。还要记住，如果当前不是作为 *db2grp1* 组的成员登录的，就无权重新启动实例！必须注销并用正确的组中的 ID 重新登录，或者将自己当前的 ID 添加进 *db2grp1* 组中。

SYSCTRL 权限

拥有 SYSCTRL 权限的用户可以在实例中执行所有管理和维护命令。但是，他们不能访问数据库中的任何数据，除非他们被授予了访问数据所需的特权。SYSCTRL 用户可以对实例中任何数据库执行的命令示例如下所示,同时 SYSCTRL 用户还包含了 SYSMAINT 用户的所有权限：

- db2 start/db2stop
- db2 create/drop database
- db2 create/drop/alter tablespace

- db2 backup/restore/rollforward database
- db2 update db cfg for database dbname

拥有 SYSADM 权限的用户可以使用以下命令将 SYSCTRL 分配给组：

```
db2 UPDATE DBM CFG USING SYSCTRL_GROUP group_name
```

SYSMAINT 权限

拥有SYSMAINT权限的用户可以发出的命令是拥有SYSCTRL权限的用户可以发出的命令的子集。SYSMAINT 用户只能执行与维护相关的任务，同时 SYSMAINT 用户还包含 SYSMON 用户的所有权限，比如：

- db2 quiesce tablespaces
- db2 reorg(针对任何表)
- db2 runstats(针对任何表)

注意：

拥有 SYSMAINT 权限的用户不能创建或删除数据库或表空间。他们也不能访问数据库中的任何数据，除非他们被显式地授予访问数据所需的特权。

SYSMON 权限

拥有 SYSMON 权限的用户可以发出与数据库监控有关的命令，比如：

- GET DATABASE MANAGER MONITOR SWITCHES
- GET MONITOR SWITCHES
- GET SNAPSHOT
- LIST (some commands):
- RESET MONITOR
- UPDATE MONITOR SWITCHES

如果拥有 SYSADM 权限，那么可以使用以下命令将 SYSMON 权限分配给组：

```
db2 UPDATE DBM CFG USING SYSMON_GROUP group_name
```

必须停止并重新启动实例，对参数的修改才会生效。

3.3.3 数据库级权限

数据库级权限包括 SECADM、DBADM、ACCESSCTRL、DATAACCESS、SQLADM、WLMADM、EXPLAIN、LOAD、CONNECT，这些权限可以分配给用户或用户组。可以

使用 GRANT 命令显式地分配这些权限。

表 3-3 总结了对用户或用户组可以授予和撤销的数据库权限类型，只有具有 SECADM 权限的用户可以授予和撤销这些权限。

表 3-3　数据库级权限总结

数据库权限	说　明
ACCESSCTRL	允许用户授予数据库内所有数据对象的访问权限(不能授予 PUBLIC)
DATAACCESS	允许用户访问数据库内的所有数据对象(不能授予 PUBLIC)
DBADM	数据库管理员
EXPLAIN	允许用户查看语句的执行计划
LOAD	允许用户使用 LOAD 实用程序加载数据
SECADM	允许用户授予和回收库内任何数据对象的所有权限(包括 DBADM)
SQLADM	控制 SQL 的执行、管理监视器
WLMADM	管理工作负载
CONNECT	允许用户连接数据库
BINDADD	允许用户在数据库中创建新的包
CREATETAB	允许用户在数据库中创建新的表
CREATE_NOT_FENCED_ROUTINE	允许用户注册定义 NOT FENCED 的用户定制函数(UDF)或存储过程
IMPLICIT SCHEMA	允许用户在尚不存在的模式中创建对象(自动地创建模式)
QUIESCE_CONNECT	允许用户连接处于 quiesced 状态的数据库
CREATE_EXTERNAL_ROUTINE	允许用户注册外部例程(用 C 和 Java 等外部语言编写的例程)

DBADM 权限

在 DB2 V9.7 中，只有 SECADM 用户才能授予和撤销 DBADM 权限，并且 DBADM 的权限被划分为 ACCESSCTRL 和 DATAACCESS 两个子权限，默认情况在给用户赋予 DBADM 权限时会自动带上这两个子权限。DBADM 是数据库级权限，而不是实例级权限。

拥有 DBADM 权限的用户可以运行如下命令：

- 创建、改变非安全相关的对象
- 读取日志文件

- 创建、激活、删除时间监视器
- 重组表
- 查询表空间状态
- 静默表空间

DBADM 用户还被自动地授予对数据库对象及其内容的所有特权。因为 DBADM 权限是数据库级权限，所以可以被分配给用户和用户组(不能是 PUBLIC 组)。以下命令演示了授予 DBADM 权限的不同方法。

```
db2 CREATE DB test
```

上面的命令将数据库 *test* 上的 DBADM 权限隐式地授予发出此命令的用户。

```
db2 CONNECT TO sample
db2 GRANT DBADM ON DATABASE TO USER xinzhuang
```

上面的命令只能由 SECADM 用户发出，它向用户 xinzhuang 授予 sample 数据库上的 DBADM 权限，默认也授予了 ACCESSCTRL 和 DATAACCESS 权限。注意，在授予 DBADM 权限之前，发出这个命令的用户必须连接到示例数据库。

```
db2 CONNECT TO sample
db2 GRANT DBADM WITH ACCESSCTRL WITHOUT DATAACCESS ON DATABASE TO USER
xinzhuang
```

上面的命令将 DBADM 并且附带 ACCESSCTRL、但不包含 DATAACCESS 的权限授予用户 xinzhuang。

```
db2 GRANT DBADM ON DATABASE to group db2grp1
```

上面的命令将 DBADM 权限授予 *db2grp1* 组中的每个用户。同样，只有 SECADM 用户能够发出这个命令。

LOAD 权限

LOAD 权限是数据库级权限，可以被分配给用户和用户组。顾名思义，LOAD 权限允许用户对表发出 LOAD 命令。当用大量数据填充表时，LOAD 命令通常用来替代插入或导入命令，速度更快。根据执行的 LOAD 操作类型的不同，有时候仅仅拥有 LOAD 权限可能还不够，可能还需要表上的特权。

拥有 LOAD 权限的用户可以运行以下命令：

- db2 load insert(必须有表上的插入特权)

- db2 load restart/terminate after load insert(必须有表上的插入特权)
- db2 load replace(必须有表上的插入和删除特权)
- db2 load restart/terminate after load replace(必须有表上的插入和删除特权)

只有拥有 SECADM 或 ACCESSCTRL 权限的用户能够对用户或用户组授予或撤销 LOAD 权限。以下示例演示了 LOAD 权限如何允许用户使用 LOAD 命令将数据装载进 *sales* 表中：

```
db2 CONNECT TO sample
db2 GRANT LOAD ON DATABASE TO USER xinzhuang
db2 GRANT INSERT ON TABLE TO USER xinzhuang
```

有了 LOAD 权限和插入特权，xinzhuang 就可以对 sales 表发出 LOAD INSERT 或 LOAD RESTART，或者在 LOAD INSERT 之后发出 TERMINATE。看下面的例子：

```
db2 GRANT LOAD ON DATABASE TO GROUP grp1
db2 GRANT DELETE ON TABLE sales TO GROUP grp1
db2 GRANT INSERT ON TABLE sales TO GROUP grp1
```

有了 LOAD 权限以及删除和插入特权，*grp1* 的任何成员就可以对 sales 表发出 LOAD REPLACE 或 LOAD RESTART，或者在 LOAD REPLACE 之后发出 TERMINATE。

3.4　特权(privilege)

3.4.1　特权层次结构

实例权限级别这种机制用于将一组预先定义的管理权限授予一组用户账号，而特权会明确分配给单独用户或组，允许他们在数据库对象上执行特定操作(例如创建和删除表)。特权给予用户通过特定方式访问数据库对象的权力，特权严格地定义了用户可以执行的任务。例如，用户可能具有读表中数据的特权，但是不能更新这些数据。图 3-7 给出了不同数据库对象的特权摘要。特权大体上分成两类：数据库特权(针对数据库中所有对象)和对象级特权(与特定对象相关联)。

图 3-7 显示了 DB2 中不同的权限和特权级别，范围涵盖了表、模式、存储过程等不同对象上的特权。

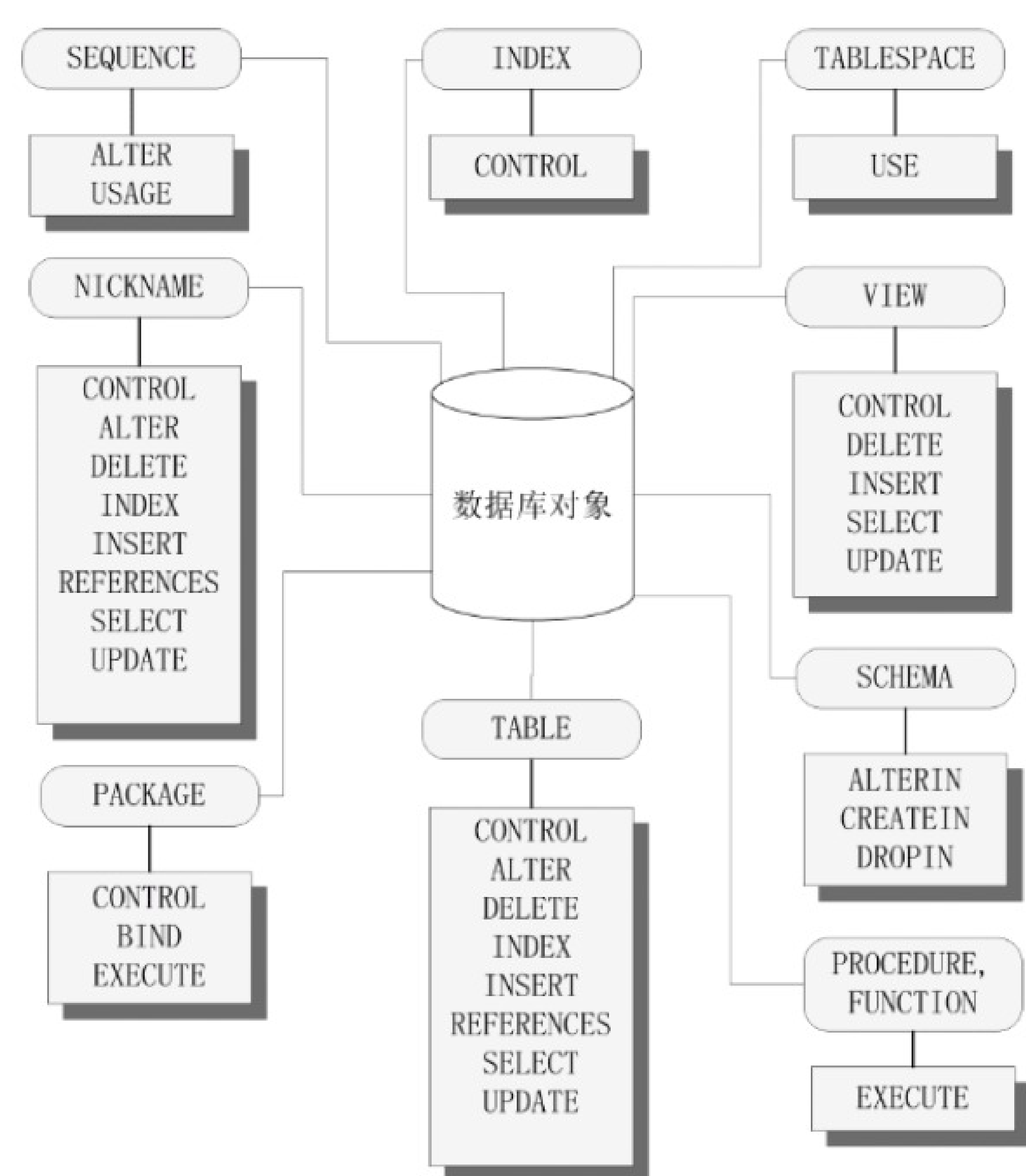


图 3-7　DB2 数据对象的特权

表 3-4 总结了对用户或用户组可以授予和撤销的唯一一种表空间特权(USE)，USE 特权不能对 SYSCATSPACE 或任何系统临时表空间使用。

表 3-4　表空间特权总结

表空间特权	说　　明
USE	允许用户在指定的表空间中创建表

表 3-5 总结了对用户或用户组可以授予和撤销的模式特权类型。

表 3-5　模式特权总结

模 式 特 权	说　　明
CREATEIN	允许用户在模式中创建对象
ALTERIN	允许用户在模式中修改对象
DROPIN	允许用户从模式中删除对象

表 3-6 总结了对用户或用户组可以授予和撤销的表/视图特权。

表 3-6　表和视图特权总结

表/视图特权	说　　明
CONTROL	授予用户在表和视图上的所有特权，以及将这些特权(除了 CONTROL)授予别人
ALTER	允许用户在表中添加列、在表和列上添加或修改注释、添加主键或唯一约束，以及创建或删除表检查约束
DELETE	允许用户从表或视图中删除行
INDEX	允许用户在表上创建索引
INSERT	允许用户在表或视图中插入数据
REFERENCES	允许用户创建和删除外键，这需要指定关系中的父表
SELECT	允许用户从表或视图中检索行，在表上创建视图以及运行 EXPORT 实用程序
UPDATE	允许用户修改表、视图，或者修改表或视图中某些列中的数据；用户可以只在特定列上具有这种特权

表 3-7 总结了对用户或用户组可以授予和撤销的唯一一种索引特权(CONTROL)。

表 3-7　索引特权总结

索 引 特 权	说　　明
CONTROL	允许用户删除索引

表 3-8 总结了对于用户或用户组可以授予和撤销的包特权类型。

表 3-8　包特权总结

包　特　权	说　　明
CONTROL	允许用户重新绑定、删除或执行包，以及将这些特权(除了 CONTROL)授予别人
BIND	允许用户重新绑定现有的包
EXECUTE	允许用户执行包

表 3-9 总结了对用户或用户组可以授予和撤销的唯一一种例程特权(EXECUTE)。

表 3-9　例程特权总结

例程特权	说　　明
EXECUTE	允许用户调用例程、从例程创建函数(只应用于函数)，以及在任何 DDL 语句(比如 CREATE VIEW、CREATE TRIGGER 或定义约束时)中引用例程

表 3-10 总结了对用户或用户组可以授予和撤销的序列特权类型。

表 3-10 序列特权总结

序 列 特 权	说 明
USAGE	允许用户对序列使用 NEXTVAL 和 PREVVAL 表达式
ALTER	允许用户使用 ALTER SEQUENCE 语句修改序列属性

3.4.2 授予特权

与实例级权限相似，可以使用命令语法授予和撤销特权。要想授予或撤销特权，必须已经连接到了数据库。以下显示了表和视图特权的 GRANT 语句的语法。其他数据库对象的 GRANT 语句语法与此相似。

```
                 .-PRIVILEGES-.
>>-GRANT--+-ALL--+------------+-------------------------+----->
          | .-,-----------------------------------------. |
          | V                                         | |
          '---+-ALTER----------------------------------+-+-'
              +-CONTROL--------------------------------+
              +-DELETE---------------------------------+
              +-INDEX----------------------------------+
              +-INSERT---------------------------------+
              +-REFERENCES--+----------------------+-+
              |             |    .-,-----------.    | |
              |             |    V             |    | |
              |             '-(----column-name-+--)-' |
              +-SELECT---------------------------------+
              '-UPDATE--+----------------------+-----'
                        |    .-,-----------.    |
                        |    V             |    |
                        '-(----column-name-+--)-'
        .-TABLE-.
>--ON--+-------+--+-table-name----+---------------------------->
                  +-view-name-----+
                  '-nickname------'

>--TO----+-+-------+--authorization-name-+-+------------------->
         | +-USER--+                       |
         | +-GROUP-+                       |
         | '-ROLE--'                       |
```

```
        '-PUBLIC------------------------'

>--+--------------------+-------------------------------------------><
   '-WITH GRANT OPTION-'
```

例如，要使用 GRANT 语句向用户 *nxz* 授予 *ACCOUNT* 表的 INSERT 特权，执行以下语句：

```
GRANT INSERT ON TABLE account TO USER nxz
```

要向 *grp1* 组授予 *CUSTOMER* 表的 SELECT 特权，执行以下语句：

```
GRANT SELECT ON TABLE customer TO GROUP grp1
```

注意：

在向用户或组授予权限和特权时必须小心，因为 DB2 允许将这些特权授予不存在的账号。如果以后创建了同名的账号，那么这个账号会自动获得以前授予的所有权限和特权。

向用户与组授予特权

从 GRANT 语句的语法图中可以看出，可以分别使用 TO USER 或 TO GROUP 子句指定是向用户还是向组授予特权。如果没有指定这两个子句之一，且指定的名称在操作系统中只定义为组，那么 DB2 会假设把特权授予 GROUP；如果指定的名称在操作系统中只定义为用户，或者没有定义，那么 DB2 会假设把特权授予 USER。如果指定的名称在操作系统中同时定义为用户和组，那么将返回错误。作为最佳实践，我们建议在 GRANT 语句中总是包含 TO USER 或 TO GROUP 子句，以避免任何二义性。

PUBLIC 组

DB2 在内部使用伪组 PUBLIC，可以对它授予和撤销特权。PUBLIC 实际上并不是外部安全设施中定义的组，而是一种向成功经过身份认证的用户分配特权的方式。可以对 PUBLIC 组授予和撤销特权，就像对其他组一样。例如，要从 PUBLIC 组撤销 IMPLICIT_SCHEMA 权限，可以发出以下语句：

```
REVOKE IMPLICIT_SCHEMA ON DATABASE FROM PUBLIC
```

重要的是，要理解向 PUBLIC 组授予特权的安全影响。任何提供了有效用户 ID 和密码的用户都能够执行 PUBLIC 组有权执行的操作。为了预防 PUBLIC 组潜在的危险，建议在创建数据库的时候指定参数 RESTRICTIVE，这样在创建数据库后，就不会自动授予 PUBLIC 任何权限。

WITH GRANT OPTION

许多数据库对象特权还允许在 GRANT 语句中包含 WITH GRANT OPTION 子句。这使你能够将一种特权授予用户或组，同时使用户或组的成员能够将同一特权授予别的用户或组。例如，以下语句将 *ACCT* 模式上的 ALTERIN、CREATEIN 和 DROPIN 特权授予组 *G1*，同时允许组 *G1* 的成员将这些特权授予别的用户或组：

```
GRANT ALTERIN, CREATEIN, DROPIN ON SCHEMA ACCT TO GROUP g1 WITH GRANT OPTION
```

WITH GRANT OPTION 并不适用于所有的 GRANT 语句，更详细的内容请参见 DB2 的 INFOCENTER。

注意：

对象的 CONTROL 权限只能由 SECADM 或 ACCESSCTRL 来授予，系统编目表上的权限也只能由这两个用户授予。并且在授予数据库级权限、索引上的权限、表或视图的 CONTROL 权限时，WITH GRANT OPTION 子句无效。

3.4.3　撤销特权

REVOKE 语句用于撤销以前授予的特权。

例如，要从用户 *JEN* 撤销 *STAFF* 表上的 ALTER 特权，可以发出以下语句：

```
REVOKE ALTER ON TABLE staff FROM USER jen
```

REVOKE 的语法如下：

```
                    .-PRIVILEGES-.          .-TABLE-.
>>-REVOKE--+-ALL--+------------+-+----ON--+-------+---------------->
           | .-,--------------.  |
           | V              |  |
           '---+-ALTER------+-+--'
               +-CONTROL----+
               +-DELETE-----+
               +-INDEX------+
               +-INSERT-----+
               +-REFERENCES-+
               +-SELECT-----+
               '-UPDATE-----'
>- ---+-table-name-+-------------------------------------------->
      +-view-name--+
      '-nickname---'
       .-,---------------------------------.
```

```
          V                              |          .-BY ALL-.
    >--FROM----+-+-------+--authorization-name-+-+--+--------+-----><
               | +-USER--+                     |
               | +-GROUP-+                     |
               | '-ROLE--'                     |
               '-PUBLIC------------------------'
```

要撤销数据库对象上的特权，必须具有 SECADM、ACCESSCTRL 权限或此对象上的 CONTROL 特权。注意，拥有 WITH GRANT OPTION 特权并不足以撤销这一特权。要从另一个用户撤销 CONTROL 特权，必须具有 SECADM 或 ACCESSCTRL 权限。

从组中的成员撤销特定特权

还有一种情况：希望将一种特权授予一个组，然后只从这个组中的某个成员撤销 CONTROL 这一特权。但是，不能撤销并未明确授予的特权。在这种情况下，有两种办法：

- 可以从这个组中删除这个成员；或者创建只包含组中其他成员的新组，并将特权授予这个新组。
- 可以从这个组撤销特权，然后向组的各个成员分别授予特权。

授予和撤销数据库权限

也可使用 GRANT 语句将数据库级权限(比如 DBADM、LOAD 和 CREATETAB)授予用户或组。例如，以下语句将 DBADM 权限授予用户 *SALLY*：

```
GRANT DBADM ON DATABASE TO USER sally
```

以下语句将 LOAD 权限授予组 *MAINT*：

```
GRANT LOAD ON DATABASE TO GROUP maint
```

如果 LOAD 操作定义为 REPLACE，那么具有 LOAD 权限的用户还需要 INSERT(为了将数据装载到表中)和 DELETE 特权。

要撤销数据库级权限，使用 REVOKE 语句。例如，要从组 *MAINT* 撤销 LOAD 权限，可以发出以下语句：

```
REVOKE LOAD ON DATABASE FROM GROUP maint
```

注意：

要撤销 DBADM 权限，必须具有 SECADM 权限。

3.4.4 显式特权/隐式特权/间接特权

显式特权

可以使用 GRANT 和 REVOKE 命令显式地对用户或组授予或撤销特权。我们来看看如何在各种对象上使用这些命令。

打开两个窗口，用具有 SECADM 权限的用户登录到系统，然后输入以下命令：

在第一个窗口中发出以下命令：

```
db2 CONNECT TO sample
```

现在，在第二个窗口中发出以下命令：

```
db2 CONNECT TO sample USER test1 USING password
```

请记住，第一个窗口中的命令是由拥有 SECADM 权限的用户发出的。第二个窗口中的命令是由 *test1* 发出的，这个用户对示例数据库没有特殊的权限或特权。注意，与示例数据库中的表相关联的模式名是发出 db2sampl 命令的用户的名称。在这些示例中，这个用户是 *gmilne*。

现在，在第二个窗口中发出以下命令：

```
db2 SELECT * FROM gmilne.org
```

应该会看到以下响应：

```
SQL0551N  "TEST1" does not have the privilege to perform operation "SELECT"
on object "GMILNE.ORG".
```

为了纠正这种状况，在第一个窗口中发出以下命令：

```
db2 GRANT SELECT ON TABLE gmilne.org TO USER test1
```

现在，前面的命令就会成功！接下来，在第二个窗口中发出如下更复杂的命令：

```
db2 INSERT INTO gmilne.org VALUES(100, 'NXZ', 1, 'NXZ', 'NXZ')
```

同样会看到错误消息：

```
SQL0551N  "TEST1" does not have the privilege to perform operation  "INSERT"
on object "GMILNE.ORG"
```

所以，在第一个窗口中输入以下命令：

```
db2 GRANT INSERT ON TABLE gmilne.org TO GROUP db2grp1
```

原来失败的 INSERT 命令现在应该会成功完成，因为 *test1* 是 *db2grp1* 组的成员。

现在，在第二个窗口中输入以下命令：

```
db2 DROP TABLE gmilne.emp_photo
```

同样会看到错误消息：

```
SQL0551N  "TEST1" does not have the privilege to perform operation "DROP
TABLE" on object "GMILNE.EMP_PHOTO".
```

所以，我们要授予这个特权。在第一个窗口中输入以下命令：

```
db2 GRANT DROPIN ON SCHEMA gmilne TO ALL
```

DROP TABLE 命令现在应该会成功完成。

既然已经完成了示例，就可以撤销刚才授予的特权。在第一个窗口中发出以下命令：

```
db2 REVOKE SELECT ON TABLE gmilne.org FROM USER test1
db2 REVOKE INSERT ON TABLE gmilne.org FROM GROUP db2grp1
db2 REVOKE DROPIN ON SCHEMA gmilne FROM all
```

注意，从组中撤销特权不一定会从这个组的所有成员撤销该特权。例如，以下命令可以用来从 *db2grp1* 撤销对 gmilne.org 表的所有特权(CONTROL 除外)：

```
db2 REVOKE ALL ON TABLE gmilne.org FROM GROUP db2grp1
```

但是，*test1* 用户(*db2grp1* 的成员)仍然拥有对这个表的 SELECT 特权，因为 *test1* 是被直接授予这个特权的。

隐式特权

当发出某些命令时，DB2 可能会自动地授予特权，而不需要像前面看到的那样发出显式的 GRANT 语句。表 3-11 总结了会导致数据库管理程序隐式地授予特权的一些命令。注意，当删除创建的对象时，这些特性会隐式地撤销。但是，当显式地撤销更高级的特权时，不会撤销这些隐式特权。

表 3-11　对于不同操作授予隐式特权的总结

操　　作	向执行此操作的用户授予的隐式特权
创建新数据库	将 DBADM、SECADM 权限以及 BINDADD、CONNECT、CREATETAB、CREATE_EXTERNAL_ROUTINE、CREATE_NOT_FENCED_ROUTINE、IMPLICIT_SCHEMA、LOAD 和 QUIESCE_CONNECT 权限授予创建者 将 BINDADD、CREATETAB、CONNECT 和 IMPLICIT_SCHEMA 授予 PUBLIC 将每个成功绑定的实用程序上的 BIND 和 EXECUTE 特权授予 PUBLIC 将系统编目表和视图上的 SELECT 特权授予 PUBLIC 将 USERSPACE1 表空间上的 USE 特权授予 PUBLIC 将 SYSFUN 模式中所有函数上的 EXECUTE WITH GRANT 特权授予 PUBLIC 将 SYSIBM 模式中所有存储过程上的 EXECUTE 特权授予 PUBLIC
授予 DBADM 权限	授予 BINDADD、CONNECT、CREATETAB、CREATE_EXTERNAL_ROUTINE、CREATE_NOT_FENCED_ROUTINE、IMPLICIT_SCHEMA、LOAD 和 QUIESCE_CONNECT
模式	在显式创建时，将 CREATEIN、ALTERIN、DROPIN 授予创建这个模式的用户 在隐式创建时，将 CREATEIN 授予 PUBLIC
创建对象(表、索引、包)	将 CONTROL 授予对象创建者
创建视图	只有在用户拥有视图定义中引用的所有表、视图和别名的 CONTROL 特权时，才授予 CONTROL 特权

例如，假设最初将 DBADM 权限授予用户 *PAUL*，以后决定撤销此权限。要从 *PAUL* 撤销 DBADM 权限，可以使用以下语句：

```
db2 REVOKE DBADM ON DATABASE FROM USER paul
```

在执行这个命令之后，*PAUL* 不再拥有 DBADM 权限；但是，他仍然拥有数据库上的 GRANT、BINDADD、CONNECT、CREATETAB、CREATE_EXTERNAL_ROUTINE、CREATE_NOT_FENCED_ROUTINE、IMPLICIT_SCHEMA、LOAD 和 QUIESCE_CONNECT 权限，这些权限是原来将 DBADM 权限授予 *PAUL* 时隐式授予的，需要从 *PAUL* 显式地撤销这些权限。

一种好的安全实践是，在创建新数据库之后，显式地撤销隐式授予 PUBLIC 的许多特权。这可以保护系统，确保只有应该访问数据库的用户才能这么做。

间接特权

当数据库管理器执行“包”时，可以间接获得特权。包中包含一条或多条 SQL 语句，这些语句已经转换为 DB2 用来在内部执行它们的格式。换句话说，包中包含可执行格式的多条 SQL 语句。如果包中的所有语句都是静态的，那么用户只需要有包上的 EXECUTE 特权，就能够成功地执行包中的语句。

例如，假设 *db2package1* 执行以下静态的 SQL 语句：

```
db2 select * from org
db2 insert into test values(1, 2, 3)
```

在这种情况下，拥有 *db2package1* 上 EXECUTE 特权的用户会间接地获得 org 表上的 SELECT 特权和 test 表上的 INSERT 特权。

3.4.5　静态和动态 SQL 特权考虑因素

如果在编译时 SQL 语句的语法是完全已知的，那么这条 SQL 语句就称为静态 SQL 语句。反之就是动态 SQL 语句，其语法直到运行时才知道。

在静态和动态 SQL 之间，处理特权的方式有一些差异。这些差异之一是如何处理组成员关系。一般来说，对动态 SQL 和非数据库对象授权(比如实例级命令和实用程序)时，需要考虑组成员关系；而静态 SQL 在执行前就已经生成了固定的执行计划并且作为数据包存放，因此不需要考虑组成员关系。这种一般性规则的一个例外发生在特权被授予 PUBLIC 时；在这种情况下，在处理静态 SQL 时也需要考虑组成员关系。

另一个差异是实际进行授权的时间。如果用静态 SQL 语句编写程序，那么必须先在数据库中创建应用程序包(包含 SQL 语句的优化执行计划)，然后相关的程序才能执行程序包中包含的 SQL 语句。对静态 SQL 的授权发生在编译或绑定时。在运行时，用户只需要有包上的 EXECUTE 特权，就能够执行其中的语句。这意味着用户不能直接访问底层数据库对象。对底层数据库对象的访问只能通过包中的特定语句进行。对于动态 SQL 语句而言，授权针对每条语句进行。执行语句的用户必须具有适当的特权，才能在运行时执行语句，特权可以是明确授予他们的，也可以是通过组成员关系授予的。

例如，假设在嵌入式 SQL 应用程序中有以下静态 SQL 语句：

```
EXEC SQL SELECT col INTO :hostvar FROM staff;
```

假设包含这条语句的应用程序文件已经预先编译了，产生了绑定文件 myapp.bnd。如果开发人员 *nxz* 希望将这个文件绑定到数据库(因此创建包)，那么她需要具有 *staff* 上的 SELECT 特权，才能成功执行 BIND 命令。这个规则有一个例外，即如果 PUBLIC 组已经被授予了这个特权，那么就不需要明确授予她这个特权。*nxz* 还需要 BINDADD 权限(如果

这是一个新的包)或 BIND 特权(假设这是一个现有的包，她希望将它重新绑定到数据库，因为数据库统计值最近更新了)。

程序包也可能包含动态 SQL，在这种情况下，需要的特权取决于在对包进行预编译或绑定到数据库时 RECOMPILE/BIND 命令的 DYNAMICRULES 子句所指定的值。如果包是使用 DYNAMICRULES RUN(默认值)绑定的，那么要想使用其中的动态 SQL，运行动态 SQL 应用程序的用户就必须具有发出每个 SQL 请求所需的特权，以及包上的 EXECUTE 特权。特权可以被授予用户的 ID、用户所在的任何组或 PUBLIC；如果应用程序是使用 DYNAMICRULES BIND 选项绑定的，那么 DB2 将包所有者的用户 ID 与应用程序包关联起来。这使运行这个应用程序的任何用户能够继承与包所有者的用户 ID 相关联的特权。

例如，假设以上例子中的应用程序文件也包含动态 SQL，如例 3-1 所示。

例 3-1 嵌入式 SQL 应用程序中的动态 SQL。

```
EXEC SQL BEGIN DECLARE SECTION;
  char hostVarStmt[50];
  short hostVarDeptnum;
EXEC SQL END DECLARE SECTION;
strcpy(hostVarStmt, "DELETE FROM org WHERE deptnum = ?");
EXEC SQL PREPARE Stmt1 FROM :hostVarStmt;
hostVarDeptnum = 15;
EXEC SQL EXECUTE Stmt1 USING :hostVarDeptnum;
```

如果 *nxz* 希望将同一绑定文件绑定到数据库，那么他需要 BINDADD 权限(如果这是一个新的包)或 BIND 特权(如果这是一个现有的包)。因此，假设 *nxz* 使用以下 BIND 命令绑定这个文件：

```
BIND sampleapp.bnd QUALIFIER u1 OWNER u2 DYNAMICRULES RUN
```

在这个例子中，所有未限定的 SQL 语句(即没有使用模式名限定其中数据库对象的语句)会使用模式 *U1*，因为使用了 OWNER 子句，用户 *U2* 会拥有这个包。因为 *nxz* 指定了 DYNAMIC RULES RUN 子句，所以会检查运行这个应用程序(执行这个包)的用户是否具有执行动态 SQL 的特权。

但是，如果 *nxz* 使用以下 BIND 命令绑定这个文件，那么会检查包的所有者是否具有执行动态 SQL 的特权：

```
BIND sampleapp.bnd QUALIFIER u1 OWNER u2 DYNAMICRULES BIND
```

在这个例子中，包的所有者被指定为 *U2*。因此，在运行时检查用户 *U2* 的特权，而不是检查运行应用程序的用户的特权。

例程特权

EXECUTE 特权应用于数据库中所有类型的例程，包括函数、存储过程和方法。用户

一旦被授予某个例程的 EXECUTE 特权，他就可以调用这个例程，从例程创建函数(只应用于函数)，以及在任何 DDL 语句(比如 CREATE VIEW 或 CREATE TRIGGER)中引用例程。对于这个例程中访问的对象来说，不需要具有对应的特权。

3.4.6　维护特权/权限

实例级权限(SYSADM、SYSCTRL、SYSMAINT 和 SYSMON)和组成员关系是在 DB2 之外定义的，因此不会反映在系统编目表中。维护实例级权限需要 root 用户和 SYSADM 组成员共同完成。

DB2 将关于特权的信息存储在 7 个系统编目视图中：

- SYSCAT.DBAUTH——数据库特权
- SYSCAT.COLAUTH——表和视图列特权
- SYSCAT.INDEXAUTH——索引特权
- SYSCAT.PACKAGEAUTH——包特权
- SYSCAT.SCHEMAAUTH——模式特权
- SYSCAT.TABAUTH——表和视图特权
- SYSCAT.TBSPACEAUTH——表空间特权

可以像查询任何其他视图一样查询这些视图。例如，要查明用户 *EMMA* 拥有哪些表特权，可以发出例 3-2 所示的命令。

例 3-2　查询 SYSCAT.TABAUTH 视图来了解特权信息。

```
    SELECT substr(grantor,1,8)AS grantor,      SUBSTR(grantee,1,8)AS grantee,
          granteetype AS gtype,       SUBSTR(tabschema,1,8)AS schema,
          SUBSTR(tabname,1,8)AS tabname,       controlauth AS ctl,
          alterauth AS alt,      deleteauth AS del,       indexauth AS idx,
insertauth AS ins,  selectauth AS sel,  refauth AS ref,      updateauth AS upd
     FROM syscat.tabauth   WHERE grantee = 'EMMA'
    GRANTOR GRANTEE  GTYPE SCHEMA   TABNAME  CTL ALT DEL IDX INS SEL REF UPD
    -------- --------- ----- --------- --------- --- --- --- --- --- --- ---
    INST1   EMMA     U    INST1    TABLE1   Y   G   G   G   G   G   G   G
```

被授权者类型(GTYPE)‘*U*’意味着被授权者(拥有此特权的账号)是用户账号，‘*G*’意味着被授权者是组账号。在其他列中，‘*Y*’意味着拥有此特权，‘*N*’意味着不拥有此特权，‘*G*’意味着拥有此特权并可以授予其他人。在例 3-2 中，可以看出用户 *EMMA* 具有表 *TABLE1* 上的 CONTROL 特权，以及所有其他可用的表特权，包括将这些特权授予其他人的能力。

注意：

隐式授予的特权——由系统授予用户的特权的授予者是 SYSIBM。

我们可以使用 AUTH_LIST_AUTHORITIES_FOR_AUTHID 表函数来查询权限(不过这个表函数的查询结果并不准确)：

```
>>-AUTH_LIST_AUTHORITIES_FOR_AUTHID---(-authid-,-authidtype-)-><
```

authid 是需要查询的用户、组、角色的名称。

authidtype 是所查询对象的类型(G 代表组、R 代表角色、U 代表用户)。

使用时请参照如下示例：

```
select * from table(AUTH_LIST_AUTHORITIES_FOR_AUTHID('xinzhuang','U'))

AUTHORITY           D_USER  D_GROUP  D_PUBLIC  ROLE_USER  ROLE_GROUP  ROLE_PUBLIC  D_ROLE
-------------       ------  -------  --------  ---------  ----------  -----------  ------
ACCESSCTRL               N        N         N          N           N            N       *
BINDADD                  N        N         Y          N           N            N       *
CONNECT                  N        N         Y          N           N            N       *
CREATE_EXTERNAL_ROUTINE  N        N         N          N           N            N       *
CREATE_NOT_FENCED_ROUTINE N       N         N          N           Y            N       *
CREATETAB                N        N         Y          N           N            N       *
DATAACCESS               N        N         N          N           N            N       *
DBADM                    Y        Y         N          N           N            N       *
EXPLAIN                  N        N         N          N           N            N       *
IMPLICIT_SCHEMA          N        N         Y          N           N            N       *
LOAD                     N        N         N          Y           N            N       *
QUIESCE_CONNECT          N        N         N          N           N            N       *
SECADM                   Y        N         N          N           N            N       *
SQLADM                   N        N         N          N           N            N       *
SYSADM                   *        Y         *          *           *            *       *
SYSCTRL                  *        Y         *          *           *            *       *
SYSMAINT                 *        Y         *          *           *            *       *
SYSMON                   *        N         *          *           *            *       *
WLMADM                   N        N         N          N           N            N       *
```

使用模式控制对数据库对象的访问

DB2 初级数据库管理员经常问的一个问题是，如何为用户创建适当的环境，让他们能够创建和删除自己拥有的数据库对象，同时限制其他用户访问这些对象。给每个用户提供自己专用的物理数据库当然可以解决这个问题，但是这不具备可行性。最优的解决方案应该是通过使用模式来控制对数据库对象的访问。

模式是一种数据库对象，用于按照逻辑将相关的数据库对象分组，还常常用来表示对象所属权。模式具有相关联的特权，使模式所有者能够控制哪些用户有权在这个模式中创建、修改和删除对象。模式所有者最初具有这个模式上的所有特权，并能够将这些特权授予其他人。具有 SECADM、ACCESSCTRL 权限的用户可以修改用户在任何模式上拥有的特权。

默认情况下，在创建数据库时，所有用户具有 IMPLICIT_SCHEMA 权限。这使任何用户可以在任何尚不存在的模式中创建对象。隐式创建的模式允许任何用户在其中创建其他对象。如果从 PUBLIC 撤销 IMPLICIT_SCHEMA 权限，那么可以使用 CREATE SCHEMA 语句明确地创建模式，而且已经被隐式授予 IMPLICIT_SCHEMA 权限的用户仍然可以隐式地创建模式。

为了让每个用户可以控制自己的数据库对象，数据库管理员可以为每个用户明确地创建模式。然后，管理员根据需要将模式上的特权授予单独的用户，这样就可以防止其他用户篡改在这个模式中创建的任何对象。为了进一步保护系统，还可以从 PUBLIC 撤销 IMPLICIT_SCHEMA 权限，这样的话，如果用户想创建数据库对象，那么必须通过具有相应特权的模式来创建。这种方式使用户能够创建他们所需要的数据库对象，同时避免篡改其他用户创建的数据库对象，或者将自己和其他人创建的对象混在一起。

注意：
模式名和用户名相同，但并不保证这个用户一定拥有这个模式相关的权限。

3.5　某银行安全规划案例

1. 任务和所需的特权/权限级别

通常，单位里的不同用户需要不同的数据库访问级别。例如，与数据库管理员相比，客户服务代表需要更受限制的访问级别。下面我们先看看有关安全的几个场景。

场景 1

小王是财务部门的一位分析师，他每天早上运行查询，查明公司的商店的收益率。在这个场景中，小王可以被授予他感兴趣的数据库上的 CONNECT 特权，以及他需要访问的所有表上的 SELECT 特权。

场景 2

小李是一位数据库管理员，负责公司中所有数据库的维护活动。她的职责包括进行备份、在需要时恢复数据库、进行存储管理并运行跟踪。她应该不能访问数据库中的任何数据。

在这个场景中，小李可以被授予 SYSMAINT 权限。如果 SYSMAINT 太受限制，也可以考虑授予 SYSCTRL 权限。

场景 3

小牛是一位应用程序开发人员，他负责开发和测试数据库管理器应用程序。他还可以

创建包含测试数据的表。

在这个场景中，小牛需要一个或多个数据库上的 BINDADD、BIND、CONNECT 和 CREATETAB，某些特定的模式特权，以及某些表上的特权。如果他用一种外部编程语言(比如 C 或 Java)开发例程，那么可能还需要 CREATE_EXTERNAL_ROUTINE。

场景 4

小刘是营销部门的一位规划师，她在每天晚上需要将从商店收集到的新数据装载进 PRODUCT_SALES 表，从而判断新的销售趋势。

在这个场景中，小刘需要数据库上的 CONNECT 特权、LOAD 权限以及 PRODUCT_SALES 表上的 INSERT 和 SELECT 特权。

2. 安全规划

从上面的例子中可以看出每个场景中需要的权限和特权都不同。必须明确每个用户或组需要完成的任务和需要的权限。我们需要做出一些安全规划。

在你的单位里，并非所有用户都以相同方式来划分工作职责。表 3-12 列示了某银行常见的安全相关的职务、与这些职务通常对应的任务以及完成这些任务需要的权限或特权。希望这个表能够有助于你规划安全。

表 3-12 常见职务、任务和必需的授权

职　务	任　务	必需的授权
部门管理员岗位	监督部门系统；创建数据库	SYSCTRL 权限。如果部门有自己的实例，那么为 SYSADM 权限
安全管理员岗位	管理一个或多个数据库中的安全性	SECADM 权限
数据库管理员岗位	设计、开发、操作和维护一个或多个数据库	对一个或多个数据库的 DBADM 和 SYSMAINT 权限。某些情况下，为 SYSCTRL 权限
系统操作员岗位	监视数据库并执行备份功能	SYSMAINT 权限
应用程序员岗位	开发和测试数据库管理器应用程序；也可以创建测试数据表	对现有程序包的 BINDADD、BIND，对一个或多个数据库的 CONNECT 和 CREATETAB，某些特定模式特权，以及对某些表的特权的列表。必需的授权
用户分析员岗位	通过检查系统目录视图来定义应用程序的数据需求	对目录视图的 SELECT；对一个或多个数据库的 CONNECT
程序最终用户	执行应用程序	对程序包的 EXECUTE；对一个或多个数据库的 CONNECT

(续表)

职　务	任　务	必需的授权
信息中心顾问	定义查询用户的数据需求；创建表和视图，并通过授予对数据库对象的访问权来提供数据	对一个或多个数据库的 DBADM 权限
查询用户	发出 SQL 语句来检索、添加、删除或更改数据；可以将结果作为表来保存	对一个或多个数据库的 CONNECT；对要创建的表有 CREATETAB，对要创建视图的模式的 CREATEIN；以及对某些表和视图的 SELECT、 INSERT、UPDATE、DELETE

3.6　执行安全审计(db2audit)

认证、权限和特权可以用来控制对数据的已知或预期存取，但是这些方法不足以阻止对数据未知或不能预测的存取。要辅助检测后面一类数据存取，DB2 提供了一种审查工具。对空闲数据访问和并发分析的成功监视可以改进对数据存取的控制，并且最后阻止恶性或无意地对数据未经授权的存取。对应用程序和个别用户存取的监视包括系统管理行为，能提供一种有关数据库系统活动的历史记录。

DB2 审查工具生成并允许 DBA 维护一系列预定义数据库事件的审计追踪。从这个工具生成的记录保存在审查日志文件中。对这些记录的分析可以揭示识别系统误用的使用模式。一旦识别出使用模式，就可以采取行动，减少或消除这些系统误用。

审计分为实例级和数据库级两种，实例级的审计意味着审计一旦开始，就会审计针对实例中所有数据库的活动；而数据库级的审计只是针对某个具体的数据库。审计功能可以监控不同对象上的不同事件，并且可以指定只记录成功的事件还是只记录失败的事件，或者两种事件都记录。

3.6.1　实例级审计

只有系统管理员(SYSADM)才能使用 db2audit 命令来配置和操作实例级的审计功能。在完成审计的配置并且生成了审计记录后，可以将审计记录提取到文本文件中，之后便可以对文本文件进行分析。还可以将审计记录提取到有分隔符的 ASCII 文件中，之后可以将该文件装载到 DB2 关系表中，以便对其进行分析和查询。db2audit 命令如下所示：

```
|>- db2audit -+- archive ---( Audit Archive )--------------------+-><|
              +- configure -+- reset -----------------+----------+
              |             '-( Audit Configuration )-'          |
```

```
            +- describe ----------------------------------------+
            +- extract -( Audit Extraction )--------------------+
            +- flush -------------------------------------------+
            +- start -------------------------------------------+
            '- stop --------------------------------------------'

Audit Archive:
>--+-----------------------------+-+-------------------------------+->
   '- database -<database name>-' '- node -+-----------------------+
                                           '-<current node number>-'

>--+-----------------------------+------------------------------------>
   '- to --<audit-archive-path>--'

Audit Configuration:
>-------+------------------------------------------------------+->
        |        .-----,-----------------------------------.  |
        |        V                                         | |
        +- scope ---+-- all -------+-- status -+- both ----+-+-+
                    +-- audit -----+           +- none ----+
                    +-- checking --+           +- failure -+
                    +-- context  --+           '- success -'
                    +-- objmaint --+
                    +-- secmaint --+
                    +-- sysadmin --+
                    '-- validate --'

>--+---------------------------+---------------------------------->
   '- errortype -+- audit --+--'
                 '- normal -'

>--+---------------------------------+---------------------------->
   '- datapath--<audit-data-path>---'

>--+------------------------------------------+--------------------->
   '- archivepath--<audit-archive-path>-----'

Audit Extraction:
>--+--------------------------------------------------------+-->
   +-file--<output file>------------------------------------+
   '-delasc--+-----------------------------+-+-----------------+
             '-delimiter-<load delimiter>-' '-to-<delasc path>-'
```

```
>--+--------------------------------------------------+----------->
   +-status--+-success-+--------------------------------+
   |         '-failure-'                               |
   |                                                  |
   |          .--,---------------------------------.  |
   |          V                                  |  |
   '-category--+-audit----+--+-------------------+-+--'
               +-checking-+  '-status-+-failure-+-'
               +-context--+           '-success-'
               +-execute--+
               +-objmaint-+
               +-secmaint-+
               +-sysadmin-+
               '-validate-'

>--from--+----------------------+--files--<input log files>-----|
         '--path-<archive path>---'
```

下面介绍每个参数的用法：

- CONFIGURE：这个参数允许在实例的安全目录中修改 db2audit.cfg 配置文件。对这个文件的更新发生在关闭实例的时候。实例激活时发生的更新动态影响着 DB2 在所有分区上进行的审查。如果已经启动了审查工具，并且正在审查可审查的事件的审查类目，那么配置文件上的配置行为就会导致创建审查记录。下面给出配置文件上可能的配置行为：
  - ◊ RESET：这个行为引起配置文件恢复初始配置，如果原始文件丢失或损坏了，就创建新的审查配置文件。
  - ◊ SCOPE：这个行为指定审查哪个类别的事件。这个行为也允许特别关注审查和减少日志的增长。推荐日志的事件类型和数目尽可能少，否则认证日志就会快速增长。审查可行的事件类目有：
    - 审查(AUDIT)：改变审查设置或存取审查日志时生成记录。
    - 认证检查(CHECKING)：试图存取的认证检查或对 DB2 对象或函数进行操作时生成记录。
    - 对象维护(OBJMAINT)：创建或删除数据对象时生成记录。
    - 安全维护(SECMAINT)：授予或撤回对象或数据库特权时生成记录。当修改数据库管理器安全配置参数 SYSADM_GROUP、SYSCTRL_GROUP 或 SYSMAINT_GROUP 时也生成记录。

- 系统管理(SYSADMIN)：当执行需要 SYSADM、SYSMAINT 或 SYSCTRL 权限的操作系统时生成记录。
- 用户确认(VALIDATE)：认证用户或再次获取系统安全信息时生成记录。
- 操作上下文(CONTEXT)：当执行数据库操作时生成记录来显示操作的上下文。这个种类允许审查日志文件更好的解释。
- 当一组事件用于日志事件的相关区域时，这组事件可以与单数据库操作相关联。例如，动态 SQL 的一条 SQL 语句、静态 SQL 的包标识符或执行的操作类型的指示器。例如，CONNECT 可以在分析审查结果时提供必要的上下文。

◊ STATUS：这个行为指定是否只有成功或失败事件被日志，抑或成功和失败事件都被日志。

◊ ERRORTYPE：AUDIT 和 NORMAL。

- DESCRIBE：这个参数描述标准输出中的当前审查配置信息和状态。
- EXTRACT：这个参数允许从审查日志移除审查记录到预示的目的地。如果没有指定可选的子句，那么所有审查记录都被提取和放到文件 reportfile 中。如果没有指定“EXTRACT”参数，那么审查记录就继续放到安全目录的 db2audit.log 文件中。如果已经存在输出文件，就返回错误消息。

下面给出用于提取的可能选项：

- FILE：提取的审查记录放到文本文件中(output_file)。
- DELASC：提取的审查记录放到限定的适于导入 DB2 关系表的 ASCII 格式文件中。

现在举一个关于审计的例子。假如，假设在某个机房维护从用户那里得到的匿名举报，说名为“王二”的用户正在试图更新他原本无权访问的数据库对象和表(存放个人工资信息的表)的访问权。于是你决定审计监控 DB2 实例，以期发现失败的授权验证尝试。

首先对审计功能进行配置，使之审计 CHECKING 事件类型，只记录失败的尝试，并且使用 NORMAL 错误处理：

```
db2audit CONFIGURE SCOPE checking STATUS failure ERRORTYPE normal
```

完成配置后，启动审计功能：

```
db2audit start
```

在审计期间，王二来到数据库服务器并完成登录。他打开命令行窗口，连接到 SAMPLE 数据库，并尝试更新 EMPLOYEE 表中的雇员工资(当然这样的尝试会遭到失败)。他发出以下 SQL 语句：

```
CONNECT TO sample USER wanger USING password
UPDATE tedwas.employee SET salary = salary * 1.5
```

于是收到以下错误消息：

```
    DB21034E The command was processed as an SQL statement because it was not
avalid Command Line Processor command. During SQL processing it returned:
    SQL0551N "wanger" does not have the privilege to perform operation "UPDATE"
onobject "TEDWAS.EMPLOYEE". SQLSTATE=42501
```

表明他没有更新那个表的许可，他快速退出服务器，并离开现场，自以为没有人注意到这一切。

一个小时过去了，你决定检查审计日志的内容，首先需要把审计日志归档：

```
$db2audit flush
AUD0000I  Operation succeeded.

$db2audit archive database test to /tmp/audit

Node     AUD      Archived or Interim Log File
        Message
-------- -------- ---------------------------------------------------
0       AUD0000I db2audit.db.TEST.log.0.20120908101017

AUD0000I  Operation succeeded.
```

然后将 db2audit.log 文件中的记录提取到带分隔符的 ASCII 文件中：

```
    db2audit EXTRACT DELASC to  /tmp/audit status failure from files
/tmp/audit/db2audit.db.TEST.log.0.20120908101017
```

为了让之前创建的 DB2 表保存审计数据，使用以下命令将从 checking.del 文件中提取的数据装载到 CHECKING 表中：

```
db2 LOAD FROM checking.del OF del INSERT INTO audit.checking
```

可以查询 AUDIT.CHECKING 表，以发现关于失败的授权尝试的更多信息：

```
SELECT category、event、appid、appname、userid、authid FROM audit.checking
```

在例 3-3 显示的查询结果中，可以看到有一条关于失败的更新语句的审计记录。

例 3-3　查询 CHECKING 表的结果。

```
SELECT category、event、appid、appname、userid、authid FROM audit.checking
```

```
CATEGORY EVENT                APPID                      APPNAME       AUTHID
---------------------- ---------------------- -------------- -------
CHECKING CHECKING OBJECT  *LOCAL.DB2.20120908101017  db2bp.exe     wanger

1 record(s)selected.
```

这个输出可以证实“王二”曾经尝试访问他无权访问的某个表。现在，你的怀疑得到了证实，可以继续收集更多的证据提交给他的领导，以便他们采取措施。

3.6.2 数据库级审计

实例级审计可以收集实例下所有数据库的审计信息，但有时我们只关心实例下某个数据库上的审计信息，这时候使用数据库级审计不仅可以减少磁盘空间浪费，也可以减轻系统压力。

为了开启数据库级审计，需要安全管理员(SECADM)权限。首先需要创建审计策略(POLICY)，审计策略用来定义我们关心的审计事件，语法如下：

```
>>-CREATE AUDIT POLICY--policy-name--●--CATEGORIES-------------->

   -,----------------------------------------------------------.
   V  (1)                                                      |
>----------+-ALL-----------------------+--STATUS--+-BOTH----+-+-->
           +-AUDIT---------------------+          +-FAILURE-+
           +-CHECKING------------------+          +-NONE----+
           +-CONTEXT-------------------+          '-SUCCESS-'
           |           -WITHOUT DATA-. |
           +-EXECUTE--+--------------+-+
           |           '-WITH DATA----' |
           +-OBJMAINT------------------+
           +-SECMAINT------------------+
           +-SYSADMIN------------------+
           '-VALIDATE------------------'

>--●--ERROR TYPE--+-NORMAL-+--●------------------------------><
                  '-AUDIT--'
```

创建完审计策略后，就可以开启数据库级的审计功能了，语法如下：

这条语句就把需要审计的对象和审计策略关联起来了，想关闭审计，只需要把审计对象上的审计策略删除即可，语法如下：

```
AUDIT TABLE audit.test USING POLICY mypolicy
```

如果想取消对象上的审计功能，删除该对象与审计策略的关联即可：

```
AUDIT TABLE audit.test REMOVE POLICY mypolicy
```

下面举一个实际中用到的例子。假如有一天，我们发现数据库中一张表里的数据莫名其妙不见了，但不知道是谁在何时删掉的，这时候就可能需要用到审计功能了。

我们先模拟用户 A 创建一张表，然后插入一些数据，紧接着创建审计策略并打开审计：

```
$db2 "create table audit.test(id int)"
$db2 "insert into audit.test values(1)"
$db2 "insert into audit.test values(2)"
$db2 "insert into audit.test values(3)"
$db2 "select * from audit.test"
ID
-----------
          1
          2
          3

  3 record(s)selected.

$db2 "create audit policy mypolicy  categories execute with data status
both error type normal"
$db2 "audit table audit.test using policy mypolicy"
$db2 "grant control on table audit.test to user db2inst1"
```

这时候我们用 db2inst1 删除表中的数据：

```
$db2 connect to test
$db2 "delete from audit.test"
$ls -l /home/db2inst1/sqllib/security/auditdata
total24 -rw - 1 db2inst1 db2iadm 9824 Sep 06 15:37 2audit.db.TEST.log.0
```

然后使用和实例级审计一样的方法，从审计日志里面抽取 ASCII 信息。上面讲过的一种方式是：首先抽取成 DEL 类型的文本文件，然后入库。这次我们将使用另外一种抽取方式，先转换成文件格式：

```
$db2audit flush
AUD0000I  Operation succeeded.

$db2audit archive database test to /tmp/audit
Node     AUD      Archived or Interim Log File  Message
-------- -------- ---------------------------------------------------
```

```
      0     AUD0000I db2audit.db.TEST.log.0.20120906154454
    AUD0000I  Operation succeeded

    $db2audit extract file /tmp/audit/audit.log from files
/tmp/audit/db2audit.db.TEST.log.0.20120906154454
    $more /tmp/audit/audit.log

      timestamp=2012-09-06-15.37.33.564018;
      category=EXECUTE;
      audit event=STATEMENT;
      event correlator=7;
      event status=0;
      database=TEST;
      userid=db2inst1;
      authid=DB2INST1;
      session authid=DB2INST1;
      origin node=0;
      coordinator node=0;
      application id=*LOCAL.db2inst1.120906073720;
      application name=db2bp;
      package schema=DB2INST1;
      package name=SQLC2H23;
      package section=203;
      local transaction id=0x0000000000003ddd;
      global transaction id=0x0000000000000000000000000000000000000000;
      uow id=2;
      activity id=1;
      statement invocation id=0;
      statement nesting level=0;
      activity type=WRITE DML;
        statement text=delete from audit.test;
        statement isolation level=CS;
        Compilation Environment Description
        isolation: CS
        query optimization: 5
        min dec div 3: NO
        degree: 1
        SQL rules: DB2
        refresh age: +00000000000000.000000
        resolution timestamp: 2012-09-06-15.37.33.000000
        federated asynchrony: 0
        maintained table type: SYSTEM;
        rows modified=3;
```

```
    rows returned=0;
    local start time=2012-09-06-15.37.33.564945;
```

可以清晰地看到 db2inst1 在 audit.test 表上执行的 delete 语句，修改了 3 行数据，执行时间，隔离级别等等。

DB2 的审计功能非常强大，可以提供审计访问尝试时所需的详细信息。虽然对未预见到的事件进行被动监控是不可避免的，但是应该使前摄性的审计成为安全计划中的重要组成部分。你还要注意安全审计对性能有些许影响。

3.7　基于标签的访问控制(LBAC)及案例

基于标签的访问控制(LBAC)为 DBA 提供了在表的行或列级限制读/写特权的能力。在以前，进行这种限制的唯一方法是创建视图，授权用户使用这个视图并撤销对基表的访问权。

基于标签的访问控制(LBAC)使安全性管理员能够准确地确定对于各行各列具有写访问权的用户和具有读访问权的用户。安全性管理员通过创建安全策略来配置 LBAC 系统。安全策略描述的是用来确定哪些用户能够访问哪些数据的条件。对于任何一个表而言，只能使用安全策略来保护它，但不同的表可以由不同的安全策略保护。

创建安全策略之后，安全性管理员将创建称为安全标签和免除权的数据库对象，这些对象是安全策略的组成部分。安全标签描述一组安全条件。免除权遵循如下规则：在拥有免除权的用户访问受安全策略保护的数据时，不需要强制对该用户比较安全标签。

一旦创建安全标签，就可以使其与各个表列和表行相关联，从而保护存放在那些位置的数据。受安全标签保护的数据称为受保护数据。安全性管理员通过将安全标签授予用户来允许该用户访问受保护数据。当用户尝试访问受保护数据时，该用户的安全标签将与用于保护该数据的安全标签进行比较。用于保护该数据的标签将阻塞一部分用户的安全标签。

LBAC 由安全管理员通过创建安全策略来设置。每个表只能由一个安全策略来控制，但是系统中可以有任意数量的安全策略。设置 LBAC 需要几个步骤。必须做的第一件事情是——决定数据需要什么类型的访问控制。下面我们举个例子。

我们做出以下假设：在你的单位有三类人，如表 3-13 所示。

表 3-13　职员及其角色

名　称	在组织中的角色
Jane	人力资源执行官
Joe	D11 和 E21 部门的经理
Frank	团队主管(A00 部门)

现在，在单位的数据库中有一个定义职员信息的表。这个表类似于 SAMPLE 数据库中的 EMP 表，包含关于职员和他们所属部门的数据。现在的定义如下：

```
db2 => describe select * from emp
 sqltype             sqllen  sqlname.data                  sqlname.length
 ------------------- ------  ----------------------------  --------------
 452   CHARACTER          6  EMPNO                                      5
 448   VARCHAR           12  FIRSTNME                                   8
 453   CHARACTER          1  MIDINIT                                    7
 448   VARCHAR           15  LASTNAME                                   8
 453   CHARACTER          3  WORKDEPT                                   8
 453   CHARACTER          4  PHONENO                                    7
 385   DATE              10  HIREDATE                                   8
 453   CHARACTER          8  JOB                                        3
 500   SMALLINT           2  EDLEVEL                                    7
 453   CHARACTER          1  SEX                                        3
 385   DATE              10  BIRTHDATE                                  9
 485   DECIMAL         9, 2  SALARY                                     6
 485   DECIMAL         9, 2  BONUS                                      5
 485   DECIMAL         9, 2  COMM                                       4
```

单位会定期对规则进行审计。审计指出，职员不应该能够访问机密的数据。规则规定，执行官对所有职员记录有完全的读/写访问权，经理对自己部门的职员记录有读/写访问权，而团队主管只能读取部门中由他们领导的职员的记录。

我们要设置 LBAC 安全策略来实现这些规则。

(1) 定义安全策略和标签，并将安全标签授予用户。

(2) 在 EMP 表中添加安全标签列并将安全策略连接到它们。

定义安全策略和标签

为了定义安全策略和标签，需要 SECADM 权限。

(1) 创建安全标签组件

首先，需要决定对于这个策略来说最合适的安全组件类型。在这个示例中，最合适的

策略类型是“TREE”。TREE 策略意味着可以定义一组标签，让子组件拥有它们父组件的权限的子集。在这个示例中，创建名为“J_DEPT”的安全组件：

```
CREATE SECURITY LABEL COMPONENT J DEPT
       TREE('HR EXECUTIVE' ROOT,
            'MAN D11 E21' UNDER 'HR EXECUTIVE'
            'A00' UNDER 'HR EXECUTIVE',
            'B01' UNDER 'HR EXECUTIVE',
            'C01' UNDER 'HR EXECUTIVE',
            'D11' UNDER 'MAN D11 E21',
            'D21' UNDER 'HR EXECUTIVE',
            'E01' UNDER 'HR EXECUTIVE',
            'E11' UNDER 'HR EXECUTIVE',
            'E21' UNDER 'MAN_D11_E21'       )
```

上面的布局说明根是 HR_EXECUTIVE，这个执行官领导的所有部门都是它的子组件。

(2) 定义安全策略

在上面的示例中，设置 LBAC 所需的下一个步骤是定义与上面的安全标签组件相关联的策略。安全策略可以使用多个组件：

```
CREATE SECURITY POLICY J DEPT POLICY
     COMPONENTS J DEPT
     WITH DB2LBACRULES
     RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
```

(3) 创建安全标签

设置安全策略的第三步是创建安全标签。在这里将指定每个用户具有的不同角色。因为这个示例非常简单，只有 3 个标签——Executive、Manager 和 Team Lead：

```
CREATE SECURITY LABEL J DEPT POLICY.EXECUTIVE
      COMPONENT J DEPT 'HR EXECUTIVE'
CREATE SECURITY LABEL J DEPT POLICY.MANAGE D11 E21
      COMPONENT J DEPT 'MAN D11 E21'
CREATE SECURITY LABEL J DEPT POLICY.A00
      COMPONENT J DEPT 'A00'
CREATE SECURITY LABEL J DEPT POLICY.B01
      COMPONENT J DEPT 'B01'
CREATE SECURITY LABEL J DEPT POLICY.C01
      COMPONENT J DEPT 'C01'
CREATE SECURITY LABEL J DEPT POLICY.D11
      COMPONENT J DEPT 'D11'
CREATE SECURITY LABEL J_DEPT_POLICY.D21
```

```
        COMPONENT J_DEPT 'D21'
CREATE SECURITY LABEL J_DEPT_POLICY.E01
        COMPONENT J_DEPT 'E01'
CREATE SECURITY LABEL J_DEPT_POLICY.E11
        COMPONENT J_DEPT 'E11'
CREATE SECURITY LABEL J_DEPT_POLICY.E21
        COMPONENT J_DEPT 'E21'
```

在下一步中，将定义与这些标签相关联的实际权限。

(4) 根据标签授予权限

下面的步骤描述了对表数据授予权限的过程。权限可以是 ALL ACCESS、WRITE ACCESS 或 READ ACCESS。如果用户没有获得上述任何权限，那么这个用户就不能访问任何表数据。请记住，执行官有完全的访问权，经理对自己的部门有完全的访问权，而团队主管只对由他们领导的部门成员有读访问权：

```
db2 grant security label J_DEPT_POLICY.A00 to user Frank for read access
db2 grant security label J_DEPT_POLICY.MANAGE_D11_E21 to user Joe for all access
db2 grant security label J_DEPT_POLICY.EXECUTIVE to user Jane for all access
```

为用户设置以上标签，根据步骤(1)中的树定义来分配权限。因为用户 Joe 被标为 MANAGE_D11_E21 并获得所有权限，所以他将能够读写那些安全标记为 J_DEPT_POLICY.D11 或 J_DEPT_POLICY.E21 的行(因为它们是子组件)。

修改 EMP 表

在修改 EMP 表时，必须创建额外的列来存储安全标签。这个列的类型是“DB2SECURITYLABEL”。可以修改 SAMPLE 数据库中现有的 EMP 表。为此，必须使用在这个策略中被授予根级特权的用户，在这个示例中就是用户 Jane：

```
CONNECT TO SAMPLE USER Jane USING password
ALTER TABLE EMP ADD COLUMN DEPT_TAG DB2SECURITYLABEL
  ADD SECURITY POLICY J_DEPT_POLICY
```

如果从 EMP 表进行选择，就会看到刚才定义的新列。由于是通过在 EXECUTIVE 级别定义的用户执行这一修改，因此添加的所有安全标记都是 EXECUTIVE。为了改变这一情况，需要更新这个表：

```
db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
varchar(SECLABEL_TO_CHAR('J_DEPT_POLICY',DEPT_TAG),30)from gmilne.emp
EMPNO  FIRSTNME     LASTNAME        WORKDEPT SALARY      6
------ ------------ --------------- --------- ----------- --------------
000010 CHRISTINE    HAAS            A00       152750.00 HR_EXECUTIVE
```

```
    000020 MICHAEL      THOMPSON       B01        94250.00 HR EXECUTIVE
    000030 SALLY        KWAN           C01        98250.00 HR EXECUTIVE
    000050 JOHN         GEYER          E01        80175.00 HR EXECUTIVE
    000060 IRVING       STERN          D11        72250.00 HR EXECUTIVE
    000070 EVA          PULASKI        D21        96170.00 HR EXECUTIVE
    000090 EILEEN       HENDERSON      E11        89750.00 HR EXECUTIVE
    000100 THEODORE     SPENSER        E21        86150.00 HR EXECUTIVE
    ..................节省篇幅，略去部分数据..................................................
    200340 ROY          ALONZO         E21        31840.00 HR EXECUTIVE
      42 record(s)selected.
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','A00'))where
WORKDEPT='A00'
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','B01'))where
WORKDEPT='B01'
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','C01'))where
WORKDEPT='C01'
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','D11'))where
WORKDEPT='D11'
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','D21'))where
WORKDEPT='D21'
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','E01'))where
WORKDEPT='E01'
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','E11'))where
WORKDEPT='E11'
    update emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','E21'))where
WORKDEPT='E21'
    db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
    varchar(SECLABEL TO CHAR('J DEPT POLICY',DEPT TAG),30)from emp
    EMPNO  FIRSTNME     LASTNAME        WORKDEPT SALARY      6
    ------ ------------ --------------- -------- -----------
    000010 CHRISTINE    HAAS           A00       152750.00 A00
    000020 MICHAEL      THOMPSON       B01        94250.00 B01
    000030 SALLY        KWAN           C01        98250.00 C01
    000050 JOHN         GEYER          E01        80175.00 E01
    000060 IRVING       STERN          D11        72250.00 D11
    ..................节省篇幅，略去部分数据..................................................
    200310 MICHELLE     SPRINGER       E11        35900.00 E11
    200330 HELENA       WONG           E21        35370.00 E21
    200340 ROY          ALONZO         E21        31840.00 E21
      42 record(s)selected.
```

在更新之后，我们来看看各个用户能够做什么。使用 EXECUTIVE 用户 ID Jane 连接数据库。首先执行与前面一样的选择语句：

```
db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
varchar(SECLABEL TO CHAR('J DEPT POLICY',DEPT TAG),30)from gmilne.emp
EMPNO  FIRSTNME     LASTNAME       WORKDEPT SALARY      6
------ ------------ --------------- -------- -----------
000010 CHRISTINE    HAAS           A00       152750.00 A00
000020 MICHAEL      THOMPSON       B01        94250.00 B01
000030 SALLY        KWAN           C01        98250.00 C01
000050 JOHN         GEYER          E01        80175.00 E01
............................节省篇幅，略去部分数据..............................................................
200330 HELENA       WONG           E21        35370.00 E21
200340 ROY          ALONZO         E21        31840.00 E21
  42 record(s)selected.
```

以及更新命令：

```
db2 => update gmilne.emp set DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','E01'))
where WORKDEPT='E01' DB20000I  The SQL command completed successfully.
```

可以看到，Jane 对表中的所有数据有完全的访问权。现在，看看 Joe 可以看到的内容。首先进行选择：

```
db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
varchar(SECLABEL TO CHAR('J DEPT POLICY',DEPT TAG),30)from gmilne.emp
EMPNO  FIRSTNME     LASTNAME         WORKDEPT SALARY      6
------ ------------ --------------- -------- -----------
000060 IRVING       STERN            D11        72250.00 D11
000100 THEODORE     SPENSER          E21        86150.00 E21
000150 BRUCE        ADAMSON          D11        55280.00 D11
000160 ELIZABETH    PIANKA           D11        62250.00 D11
000170 MASATOSHI    YOSHIMURA        D11        44680.00 D11
000180 MARILYN      SCOUTTEN         D11        51340.00 D11
000190 JAMES        WALKER           D11        50450.00 D11
000200 DAVID        BROWN            D11        57740.00 D11
000210 WILLIAM      JONES            D11        68270.00 D11
000220 JENNIFER     LUTZ             D11        49840.00 D11
000320 RAMLAL       MEHTA            E21        39950.00 E21
000330 WING         LEE              E21        45370.00 E21
000340 JASON        GOUNOT           E21        43840.00 E21
200170 KIYOSHI      YAMAMOTO         D11        64680.00 D11
200220 REBA         JOHN             D11        69840.00 D11
200330 HELENA       WONG             E21        35370.00 E21
200340 ROY          ALONZO           E21        31840.00 E21
  17 record(s)selected.
```

看到了吗？他只能看到 D11 和 E21 部门的信息。如果试图选择不允许他访问的表数据，看看会发生什么：

```
    db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
    varchar(SECLABEL TO CHAR('J DEPT POLICY',DEPT TAG),30)
    from gmilne.emp where empno='000130'
    EMPNO  FIRSTNME     LASTNAME         WORKDEPT SALARY      6
    ------ ------------ ---------------- --------- -----------
      0 record(s)selected.
```

在前面 Jane 进行选择的结果中我们看到，有个职员的 empno 是 000130，但是现在却不允许 Joe 看到它。

下面是最后一个测试，对用户 Frank 的测试。

首先，运行与前两个用户相同的选择：

```
    db2 => select EMPNO, FIRSTNME, LASTNAME, WORKDEPT, SALARY,
    varchar(SECLABEL TO CHAR('J DEPT POLICY',DEPT TAG),30)from gmilne.emp
    EMPNO  FIRSTNME     LASTNAME        WORKDEPT  SALARY      6
    ------ ----------- ---------------- --------- -----------
    000010 CHRISTINE    HAAS            A00       152750.00 A00
    000110 VINCENZO     LUCCHESSI       A00        66500.00 A00
    000120 SEAN         O'CONNELL       A00        49250.00 A00
    200010 DIAN         HEMMINGER       A00        46500.00 A00
    200120 GREG         ORLANDO         A00        39250.00 A00
      5 record(s)selected.
```

在这里可以看到，Frank 只能看到部门中由他领导的用户的相关信息。我们来看看在他尝试进行更新时会发生什么：

```
    db2 => update gmilne.emp set
DEPT TAG=(SECLABEL BY NAME('J DEPT POLICY','A00'))
    where WORKDEPT='A00'DB21034E  The command was processed as an SQL statement
    because it was not a valid Command Line Processor command.  During SQL
processing it
    returned:
    SQL20402N AuthorizationID"FRANK" does not have the LBAC credentials to
    perform the "UPDATE" operation on table "EMPLOYEE".  SQLSTATE=42519
```

尽管尝试更新的记录是在他自己的部门中，但是访问安全策略只允许他对表进行读访问。可以看出，我们的业务需求已经得到了满足。

3.8 本章小结

在本章我们讲解了 DB2 中定义的各种权限级别和特权，以及如何使用命令行语法和控制中心将它们分配给用户，还讨论了特权的某些细节，包括隐式特权、静态和动态 SQL 之间授权方式的差异，以及特权信息如何存储在系统编目表中。最后，讨论了在多用户环境中如何使用模式有效地控制对数据库对象的访问、如何有效地规划安全。有了这些知识后，就应该能够定义特权/权限策略，防止用户意外或故意地威胁系统的安全。

第 4 章 OLTP 系统设计与管理

OLTP 系统，也叫联机事务处理(Online Transaction Processing)系统，指的是以小交易、高并发为主的系统，每秒执行的事务数可以高达上千笔，单个事物执行时间很短，一般在毫秒级。这种类型的系统主要存在于各交易系统(比如银行的存取款系统、电信的计费系统等)、电子商务系统(比如淘宝、亚马逊等)中，这种系统往往都非常重要，一旦出现问题，就会对企业造成非常大的损失。因此对于这种系统，我们要对其负载量、响应时间等有充分的评估，在硬件环境选择上要留出充分的缓冲，在基础设置、物理设计、代码开发上要有充分的优化。

本章主要讲解如下内容：

- 基础环境设计
- 物理设计
- 数据库对象的设计原则
- 代码开发的基本原则

4.1 基础环境设计

经济学上讲，“经济基础决定上层建筑”，说的是只有好的物质基础，才能发展好的经济形态、社会关系、政治、法律、哲学、宗教、艺术等，没有好的物质基础，其他一切都是空谈。同理，在数据库设计上，底层设计决定上层应用，没有好的基础设计，上层应用设计再好、再花哨也发挥不出来。基础环境设计主要包括硬件环境设计、操作系统设计、

实例和数据库参数设置、物理结构设计等。

4.1.1　硬件环境设计

由于 OLTP 系统具有并发大、延时短的特点，因此在硬件上要选择多核、高主频的服务器，比如 IBM 的 Power 8 处理器或 x86 的安腾处理器等。服务器的网卡尽量选择处理能力更强、延时更短的万兆网卡。本地硬盘建议选择 SAS 或 SSD 硬盘。存储要选择转速高的磁盘，存储的机头要具有较大的缓冲和较快的处理性能。存储的磁盘数量要足够多，防止出现热点磁盘。硬件环境的细节部分在《DB2 数据库性能调整和优化(第 3 版)》的第 2 章进行了详细阐述，这里不再展开。

4.1.2　操作系统设计

操作系统要选择较新的版本，建议选择 Linux 或 AIX，新版本的 Linux 或 AIX 都支持 CIO 的模式，在性能上不亚于裸设备。CIO 和裸设备的细节部分也在《DB2 数据库性能调整和优化(第 3 版)》的第 2 章进行了详细阐述，这里不再展开。

4.1.3　实例和数据库参数设置

根据笔者自己多年的维护和调优经验，总结了一些常用的参数设置，推荐给大家。虽然不是最好的，但是配置后可以应付绝大多数问题。OLTP 系统的实例级参数可以在 db2set 和 dbm cfg 这两个地方配置，db2set 里参数的推荐设置如表 4-1 所示：

表 4-1　db2set 推荐参数

NAME	VALUE	原因
DB2_EVALUNCOMMITTED	ON	提高锁的性能
DB2_SKIPINSERTED	ON	提高锁的性能
DB2COMM	TCPIP	打开 TCP/IP 连接
DB2_PARALLEL_IO	*	所有表空间都将使用 6 作为每个容器的磁盘数
DB2_USE_IOCP	ON	启用异步 I/O
DB2_SQLWORKSPACE_CACHE	500	加大 SQL 工作空间
DB2FODC	DUMPCORE=OFF	不允许 COREDUMP
DB2_OPTPROFILE	YES	启用 optimization profile
DB2_APM_PERFORMANCE	16	减少 package cache 的锁竞争

dbm 的推荐参数如表 4-2 所示：

表 4-2　dbm 推荐参数

NAME	VALUE	原因
DFT_MON_BUFPOOL	ON	打开缓冲池的监控指标
DFT_MON_LOCK	ON	打开锁的监控指标
DFT_MON_SORT	ON	打开排序的监控指标
DFT_MON_TABLE	ON	打开表的监控指标
DFT_MON_TIMESTAMP	ON	打开时间戳
DFT_MON_UOW	ON	打开事务的监控指标
DFT_MON_STMT	ON	打开语句的监控指标
HEALTH_MON	OFF	关闭健康监控
INSTANCE_MEMORY	固定值	一般设置为系统可用内存的 60%
SVCENAME	端口	设置 TCP 的服务端口
SYSMON_GROUP	组名	设置监控组
DIAGPATH	独立路径	设置 DB2 诊断日志的路径
AUTHENTICATION	SERVER_ENCRYPT	设置使用认证模式

数据库级别的推荐参数如表 4-3 所示：

表 4-3　数据库级别的推荐参数

NAME	VALUE	原因
DATABASE_MEMORY	固定值	设置数据库可用的内存空间
LOGFILSIZ	20480 或更大	单个日志文件的大小
LOGPRIMARY	50	主日志的文件个数
LOGSECOND	100	辅助日志的文件个数
LOGBUFSZ	1024	日志缓冲区大小
LOGARCHMETH1	独立路径	启用归档日志
LOCKTIMEOUT	30	锁超时的时间(秒)

(续表)

NAME	VALUE	原因
MAX_LOG	60	每个事务可以占用的主日志空间的百分比
PKGCACHESZ	32768	程序包高速缓存大小
NEWLOGPATH	独立路径	活动日志的路径
REC_HIS_RETENTN	60	指定保留备份历史信息的天数
MON_LCK_MSG_LVL	3	关于锁定升级、死锁和锁定超时的通知
AUTO_STMT_STATS	OFF	关闭实时统计分析
MON_LW_THRESH	5000000	触发锁等待抓取信息的时间阈值
MON_LOCKTIMEOUT	WITHOUT_HIST	监视锁超时时不记录历史
MON_LOCKWAIT	WITHOUT_HIST	监视锁等待时不记录历史

4.2　物理结构设计

4.2.1　DB2 页大小的选择

数据页是 DB2 数据库进行磁盘 I/O 操作的最小单位。在创建数据库时我们可以通过 PAGESIZE 设定页大小，不指定的情况下默认为 4KB。在 DB2 数据库中目前支持的数据页大小为 4KB、8KB、16KB、32KB。

要确定表空间的数据页大小，必须考虑下列事项：

- 对于执行随机行读写操作的 OLTP 应用程序，通常最好使用较小的页大小，这样不需要的行浪费的缓冲池空间就会较少。另外，较小的页大小可以更好地打散数据，降低热点数据的可能。
- 对于一次访问大量连续行的 OLAP 应用程序，页大小大一些会比较好，这样就能减少读取特定数目的行所需的 I/O 请求数。但是，也有例外。如果行大小小于 pagesize / maximum rows，那么在每一页上都会浪费空间。在这种情况下，更小一点的页大小可能更合适。
- 更大的页大小可允许减少索引中的级别数。越大的页，支持的行越长。如果使用默认页大小(4KB)，表最多可以有 500 列。较大的页大小(8KB、16KB 和 32KB)支持 1012 列。表空间的最大大小与表空间的页大小成正比。

总之，在满足以上条件的情况下，交易系统使用较小的页更适合，仓储系统使用较大的页更适合。

4.2.2　表空间类型的选择

系统管理的空间(System-Managed Space，SMS)：在这里，由操作系统的文件系统管理器分配和管理空间。在 DB2 V9 之前，如果不带任何参数创建数据库或表空间，就会导致所有表空间作为 SMS 对象创建。这种表空间依赖底层的操作系统(例如 AIX、HP-UX 或 Windows)来进行空间管理。

数据库管理的空间(Database-Managed Space，DMS)：在这里，由 DB2 数据库管理程序控制存储空间。表空间容器可使用文件系统或裸设备。

DMS 的自动存储(Automatic Storage With DMS)：自动存储实际上不是一种单独的表空间类型，而是一种处理 DMS 存储的不同方式，这种类型的表空间可以减少系统管理员的维护工作量，同时保持 DMS 表空间的高性能，因此建议在生产和开发环境中采用。

4.2.3　页大小、表大小和表空间大小

数据库的页大小决定了数据库的表空间大小，其对应关系如表 4-4 所示。

表 4-4　表空间和页大小的对应关系

表空间类型	4KB 页大小限制	8KB 页大小限制	16KB 页大小限制	32KB 页大小限制
SMS 表空间	64	128	256	512
DMS 表空间(常规)	64	128	256	512
DMS 表空间(常规)	2048	4096	8192	16384
自动存储器表空间(大型)	64	128	256	512
自动存储器表空间(大型)	2048	4096	8192	16384
临时表空间	64	128	256	512

因此在选择页大小的时候要充分考虑表空间大小的限制，并且在创建存放用户数据的表空间时，要选择 LARGE 类型的表空间。如果数据库太大，可以考虑采用多个表空间存放或者采用分区表或分区数据库的技术。

4.2.4　表空间参数的设置

在一个数据库内创建表空间，会将容器分配到表空间，并在数据库系统目录中记录它的定义和属性。我们需要根据需求、性能、存储数据量等考虑选择什么样的表空间(SMS、DMS)？需要创建哪些表空间(large tablespace、temporary tablespace...)？

在较为正式的测试环境和正式应用场景下，建议选择创建 DMS 表空间，性能会较 SMS 好一些；另外，根据数据的平均长度选择合适的数据页也非常重要。

建议将表的数据和索引单独存放在不同的表空间，这样方便管理，同时可以单独缓存，对性能也有一定的好处。

要使用命令行来创建 SMS 表空间，请输入：

```
CREATE TABLESPACE name
MANAGED BY SYSTEM
USING ('path')
```

要使用命令行来创建 DMS 表空间，请输入：

```
CREATE TABLESPACE name
MANAGED BY DATABASE
USING (FILE 'path' size)
```

注意：

默认情况下，DMS 表空间将被创建为大型表空间。

要使用命令行创建自动存储器表空间，请输入下列任一语句：

```
CREATE TABLESPACE  name
--或者
CREATE TABLESPACE name
MANAGED BY AUTOMATIC STORAGE
```

假定在自动存储器数据库中创建表空间，以上两条语句是等同的；默认情况下，除非另有指定，否则在此类数据库中创建的表空间将是自动存储器表空间。

在创建表空间时需要合理地定义这些参数，如下示例展示了创建表空间的方法，稍后会解释具体的参数。

```
CREATE LARGE TABLESPACE ODS DAT 32
PAGESIZE 32K
MANAGED BY DATABASE
USING  (FILE '/db2/db2xb2/ ODS DAT 32'  10G)
EXTENTSIZE 32
PREFETCHSIZE AUTOMATIC
BUFFERPOOL BP DATA 32
NO FILE SYSTEM CACHING;
```

- PAGESIZE：指定表空间的页大小。
- MANAGED BY DATABASE：表明表空间由数据库管理，非自动扩展。
- EXTENTSIZE：指明表空间内部逻辑块扩展的大小，建议在 OLTP 环境中设置为 32。

- PREFETCHSIZE：指定执行数据预取时将从表空间中读取的预取页数，在 DB2 中，PREFETCHSIZE 的值可以设置为表空间容器数、每个容器下的物理磁盘数(如果使用了 RAID 设备的话)与表空间的 EXTENTSIZE 值(数据库管理器在使用另一个容器前写入容器的页数)的乘积。这里我们建议设置为 AUTOMATIC。
- BUFFERPOOL：指定表空间使用的 BUFFERPOOL，缓冲池必须和表空间页大小设置相同。
- NO FILE SYSTEM CACHING：指定表空间所使用的容器不经过操作系统缓存，这样可以提高性能。

4.2.5　数据库 BUFFERPOOL 的创建和设置

缓冲池为数据库页提供工作内存和高速缓存。

缓冲池通过允许从内存(而不是磁盘)中读取数据来提高数据库系统的性能。由于大多数页数据处理发生在缓冲池内，因此配置缓冲池是唯一最为重要的调整环节。

当应用程序访问表行时，数据库管理器将在缓冲池中查找包含该行的页。如果在缓冲池中找不到该页，那么数据库管理器将从磁盘中读取该页并将其放入缓冲池。然后，可以使用该数据来处理查询。

创建缓冲池时，除非显式指定另一页大小，否则页大小将是创建数据库时指定的大小。由于仅当表空间页大小与缓冲池页大小相同时才可以将页读入缓冲池，因此对缓冲池指定的页大小应该由表空间的页大小确定。在创建缓冲池后，就无法更改它的页大小。

可以使用如下语句查看当前数据库有哪些 BUFFERPOOL：

```
    db2 "select substr(BPNAME,1,30) as BPNAME,NPAGES,PAGESIZE from
SYSCAT.BUFFERPOOLS"
```

在 DB2 数据库中不同页大小的表空间至少对应一个页大小相同的 BUFFERPOOL。在生产环境中，建议索引和数据的缓冲池分开、业务数据和字典数据的缓冲池分开、热点表的缓冲池分开。

以下是缓冲池创建语句的一个示例：

```
    CREATE BUFFERPOOL  <BP_NAME>  SIZE 327680  PAGESIZE 16K;
```

- SIZE：指定缓冲池的页数。
- PAGESIZE：指定缓冲池的页大小。

4.3　数据库对象的设计原则

4.3.1　表相关的设计原则

1. 对大对象类型的操作

LOB 是存储大量数据的数据库字段，例如图形文件或长的文本形式的文档。DB2 提供了三种不同类型的 LOB：CLOB、BLOB、DBCLOB，参见表 4-5。

表 4-5　LOB 类型

LOB 类型	说明
CLOB	用于存储单字节字符数据
BLOB	存储没有结构的二进制数据
DBCLOB	用于存储双字节字符数据

LOB 用于存储非结构化的数据。所谓非结构化数据，是指不能被分解为标准组件的数据。例如，一名员工可以分解为姓名(字符串)、标识(数字)、薪水等。但是，如果给的是一张图片，会发现数据是由一长串 0 和 1 组成的数字流，在数据存储方面，它们不能分解为良好的结构。非结构化的数据往往是非常大的，如文本、图片、视频片段或音乐等。一条典型的员工记录可以有几百个字节，但即使是少量的多媒体数据，也可能有它几千倍那么大。

LOB 类型帮助支持 Internet 应用，随着 Internet 和内容丰富应用的发展，数据支持这样一种数据类型是非常有需求的：1) 可以存储非结构化数据；2) 优化处理大量的非结构化数据；3) 为存储在数据库内或数据库外的非结构化数据提供统一的访问方式。

通常，大对象(LOB)与引用它们的表行存储在不同的位置。但是，可以选择以直接插入方式将长度不超过 32 673 字节的 LOB 存储在基本表行中，以便简化对其进行的访问。

将大数据对象存储在基本表行中可能并不现实(根据数据的不同，可能无法进行此存储)。图 4-1 提供了尝试将 LOB 存储在表行中的示例，并说明了为何这样做会引起问题。在此例中，行被定义为包含两个 LOB 列，其长度分别为 500 和 145 千字节。但是，DB2 表的最大行大小是 32KB；因此，这样的行定义实际上根本无法实现。

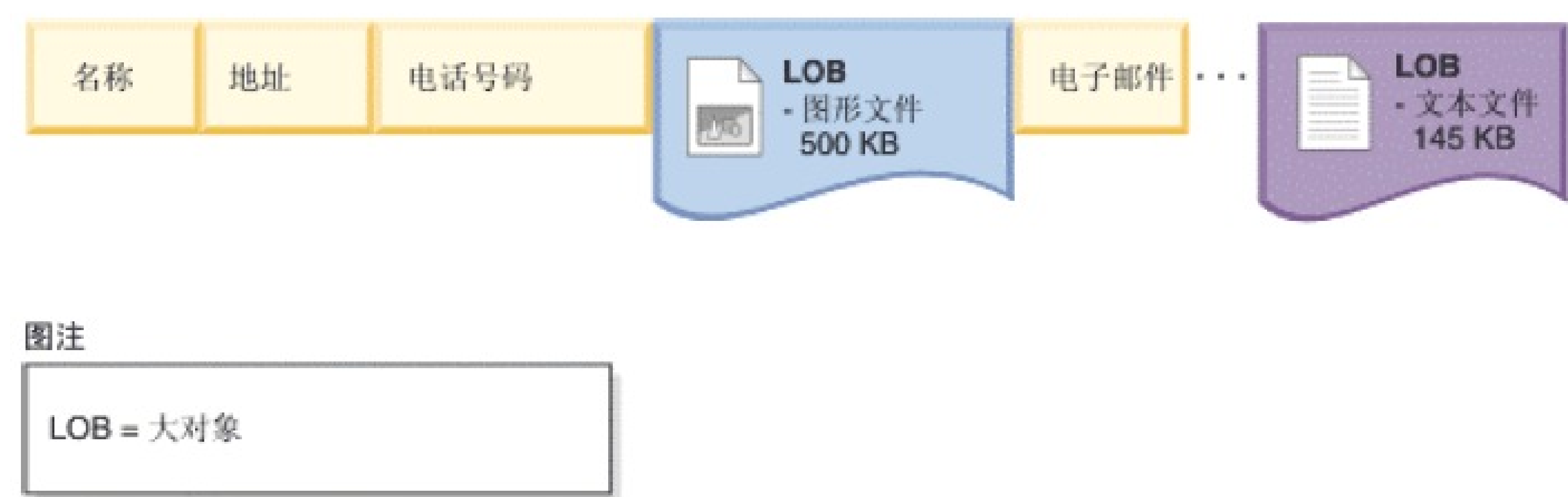


图 4-1　将 LOB 数据存储到基本表行时存在的问题

为了降低使用 LOB 时的难度，处理它们的方式与处理其他数据类型不同。图 4-2 显示只将 LOB 描述符(而不是 LOB 本身)存储在基本表行中。每个 LOB 本身都存储在数据库管理器所控制的另一 LOB 位置。采用这种安排方式后，包含 LOB 描述符的行与包含完整 LOB 的行相比，将缩短在缓冲池与磁盘存储器之间移动它们所需的时间。但是，由于实际的 LOB 与基本表行存储在不同的位置，因此处理 LOB 数据变得更为困难。

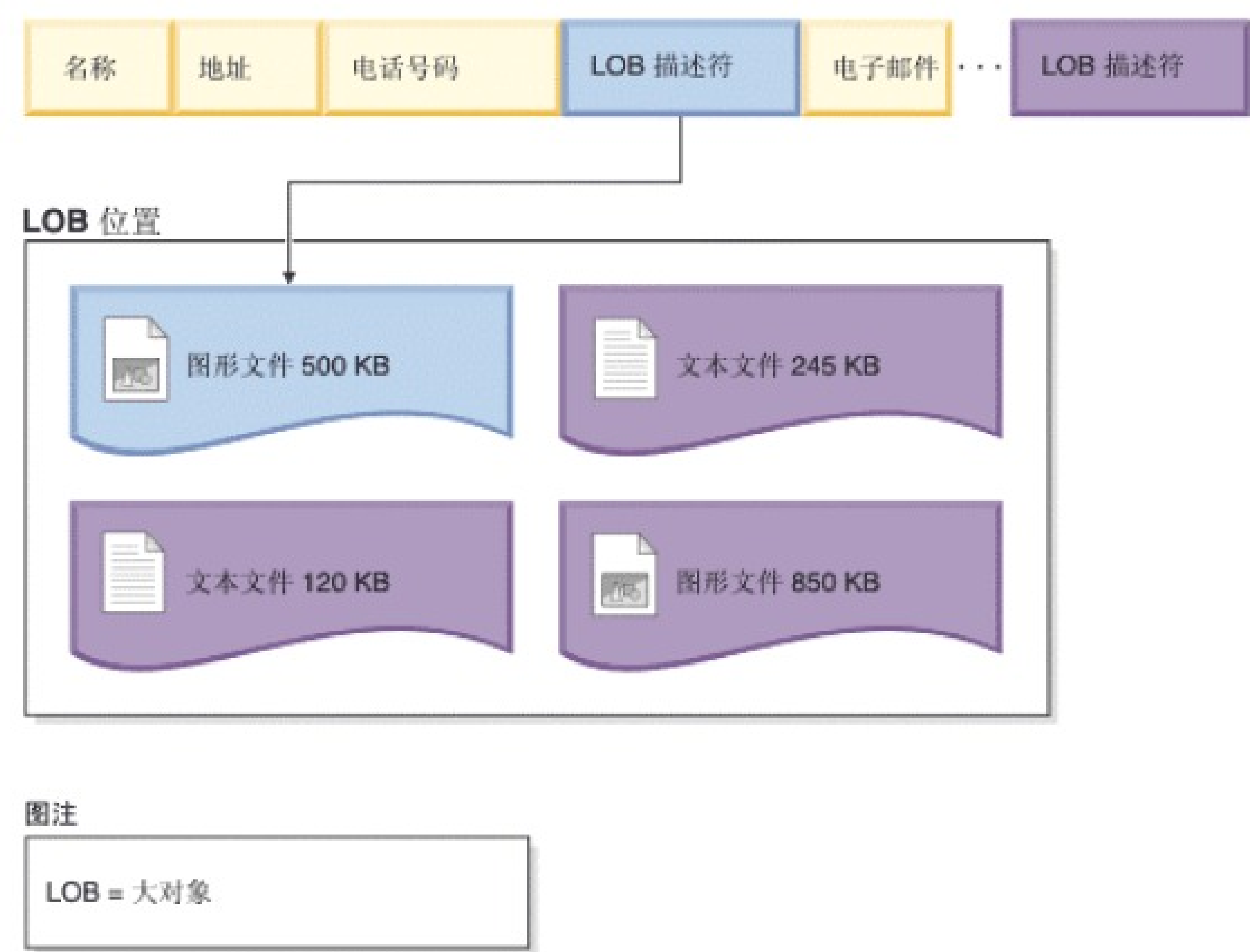


图 4-2　基本表行中的 LOB 描述符引用独立 LOB 位置的 LOB

为了存储一些二进制数据或文件数据，可以在表中使用 LOB 类型的字段，为了提升 LOB 数据处理的性能和独立性，可以在 LOB 所属表的定义中指定单独的表空间，指定时使用关键字 LONG IN <tablespace>。

在 DB2 V9.7 及以后的版本中，在 LOB 较小时(指小于表空间页大小)，基于以下两个目的，可以选择使用 INLINE 的方式存放 LOB 数据。

- 对 LOB 数据的并发处理较大的情况下，如几十个并发处理 LOB 数据。
- LOB 占用空间较大，需要压缩其空间占用的情况下。

对于没有 INLINE 的 LOB 类型，DB2 使用直接读写的方式进行操作，每次 I/O 操作都

要写入磁盘，这会大大降低读写性能。用了 INLINE LENGTH 的字段可以具有 VARCHAR 类型一样的性能，既可以被缓冲池缓存，也可以被 DB2 压缩，如图 4-3 所示。



图 4-3　存储在基本表行中的小型 LOB

可以在使用 CREATE TABLE 或 ALTER TABLE 语句的时候声明 INLINE LENGTH 选项来提供这一功能，INLINE LENGTH 指定的大小不能超过表空间页大小的定义。例如：

```
CREATE TABLE TEST (
C1 int,
C2 CLOB INLINE LENGTH 2048)
```

C2 字段里小于 2048 字节的数据都会被 INLINE 处理。但是，如果 C2 里有些行超过了 2048 字节，则不会被 INLINE，而是被直接写入 LOB 区。

DB2 提供了两个系统函数来辅助我们实施 INLINE 操作，分别是 ADMIN_EST_INLINE_LENGTH 和 ADMIN_IS_INLINED，它们分别用来评估 INLINE 需要的长度和是否已经 INLINE，例如：

```
db2 => SELECT PK, ADMIN_IS_INLINED(xml_doc1) as IS_INLINED,
              ADMIN_EST_INLINE_LENGTH(xml_doc1) as EST_INLINE_LENGTH
       from TAB1
```

执行结果如下：

```
PK          IS_INLINED EST_INLINE_LENGTH
----------- ---------- -----------------
          1          1               292
          2          0               450
          3          0               454

  3 record(s) selected.
```

第一行记录已经 INLINE，所以 EST_INLINE_LENGTH 返回该行的实际长度；第二

行和第三行没有 INLINE，所以 EST_INLINE_LENGTH 返回该行需要设置的 INLINE 的长度。

2. 自增列和序列对象

为了方便用户产生有序自增的数值，比如 1、2、3、4……，DB2 提供了自增列(IDENDITY)和序列对象(SEQUENCE)。自增列被直接定义为列的属性，使用起来比较简单，但是维护相对复杂，性能不高，灵活性也不够。相比自增列，序列对象提供了更大的灵活性和更好的性能，因此笔者建议大家尽量使用序列对象来实现自增的数值。创建序列对象的语法如下：

```
>>-CREATE--+------------+--SEQUENCE--sequence-name--●----------->
           '-OR REPLACE-'

   .-AS INTEGER--------.
>--+-------------------+--●--+------------------------------+--->
   '-AS--| data-type |-'     '-START WITH--numeric-constant-'

      .-INCREMENT BY 1-----------------.
>--●--+--------------------------------+--●--------------------->
      '-INCREMENT BY--numeric-constant-'

   .-NO MINVALUE----------------.
>--+----------------------------+--●---------------------------->
   '-MINVALUE--numeric-constant-'

   .-NO MAXVALUE----------------.     .-NO CYCLE-.
>--+----------------------------+--●--+----------+--●----------->
   '-MAXVALUE--numeric-constant-'     '-CYCLE----'

   .-CACHE 20----------------.     .-NO ORDER-.
>--+-------------------------+--●--+----------+--●-------------><
   +-CACHE--integer-constant-+     '-ORDER----'
   '-NO CACHE----------------'

data-type

|--+-| built-in-type |------+-----------------------------------|
   |                   (1)  |
   '-distinct-type-name-----'

built-in-type
```

```
|--+-+-SMALLINT----+-------------------------------+-------------|
   | +-+-INTEGER-+-+                               |
   | | '-INT-----' |                               |
   | '-BIGINT------'                               |
   |                .-(5,0)-------------------. |
   '-+-+-DECIMAL-+-+--+-----------------------+-'
     | '-DEC-----' |  |         .-,0-------.   |
     '-+-NUMERIC-+-'  '-(integer-+----------+-)-'
       '-NUM-----'              '-,integer-'
```

创建过程非常简单，只需要注意以下几点即可：

1) 要注意序列的最大值，笔者就亲历过上线的生产系统因序列达到最大值而造成服务不可用的情况。表 4-6 列出了不同类型的序列对应的最大值。

表 4-6　不同类型序列的最大值

类　型	最　大　值
SMALLINT	32768
INTEGER	2147483648
BIGINT	9223372036854775808

2) 要注意使用缓冲。默认 CACHE 为 20，在高并发的 OLTP 环境中，20 根本不够，会造成大量的 latch 竞争，因此笔者建议大家调整为 1000。

3) 选择循环模式的话，要注意数据冲突问题。选择 CYCLE 属性的话，序列达到最大值后会从头开始循环，这时一定要注意不要和老的数据有冲突。

在这里向大家介绍数据库中序列或自增列达到最大值后的解决方法。比如在使用序列或自增列的过程中，出现以下报错：

```
SQL0359N  The range of values for the identity column or sequence is
exhausted.  SQLSTATE=23522
```

当出现 SQL0359N 时，表明数据库中序列或自增列达到了最大值，并且序列或自增列采用的是 NO CYCLE 方式。

具体来说有两种可能：

- 最大值是手动指定的、定义的太小。
- 序列值采用的数据类型不恰当，最大值采用的是该数据类型的最大值。

我们推荐五种思路来解决这种问题，具体使用哪种可以根据系统自行选择。

1) 最直接的思路，是将序列或自增列重置。

2) 更改最大值的大小。
3) 更改为循环使用的方式。
4) 更改数据类型。
5) 如果该表存的数据是临时数据，可以将该表重命名，然后重建。

利用下面的脚本，确定哪些序列或哪些表的自增列存在问题。

备注：该脚本列出了序列当前值大于最大值 50%以上的序列或表的自增列。

```
    select substr(a.TABSCHEMA,1,12) TABSCHEMA, substr(a.TABNAME,1,20) TABNAME,
substr(a.COLNAME,1,12) COLNAME, substr(b.typename,1,12) type,
a.NEXTCACHEFIRSTVALUE,a.MAXVALUE, a.CYCLE from syscat.COLIDENTATTRIBUTES a,
syscat.columns b where a.TABSCHEMA not like 'SYS%' and a.tabschema=b.tabschema
and a.tabname=b.tabname and a.colname=b.colname and a.cycle='N' and
(a.NEXTCACHEFIRSTVALUE > 0.5*a.MAXVALUE or a.NEXTCACHEFIRSTVALUE is null) order
by a.NEXTCACHEFIRSTVALUE desc with ur

    select substr(a.SEQSCHEMA,1,12) SEQSCHEMA,substr(a.SEQNAME,1,20) SEQNAME,
substr(a.owner,1,12) OWNER,substr(b.typename,1,12) type,
a.NEXTCACHEFIRSTVALUE,a.MAXVALUE, a.CYCLE from syscat.sequences a,
syscat.datatypes b where a.SEQSCHEMA not like 'SYS%' and a.datatypeid=b.typeid
and a.cycle='N' and a.SEQTYPE='S' and (a.NEXTCACHEFIRSTVALUE > 0.5*a.MAXVALUE
or a.NEXTCACHEFIRSTVALUE is null) order by a.NEXTCACHEFIRSTVALUE desc with ur
```

第一种办法：将序列或自增列重置，前提是该列不是主键或存在唯一索引。

自增列：

```
    alter table xxx.xxx alter column xxx restart with 1
```

序列：

```
    alter sequence xxx restart with 1
```

第二种办法：更改最大值的大小，前提是最大值是自定义的，小于该数据类型支持的最大值。

自增列：

```
    alter table xxx.xxx alter column xxx set maxvalue xxx
```

然后查看表状态是否为 normal。

```
    load query table xxx
```

序列：

```
alter sequence xxx maxvalue xxx
```

第三种办法：更改成循环使用的方式。

自增列：

```
alter table xxx.xxx alter column xxx set cycle
```

然后查看表状态是否为 normal。

```
load query table xxx
```

序列：

```
alter sequence xxx cycle
```

第四种办法：更改数据类型，不建议采用，因为这会导致表处于 reorg pending 状态，需要进行 reorg。根据测试结果，自增列不能直接修改数据类型，需要删除属性，更改数据类型，添加属性，然后 reorg。即使这样，自增列的取值还是会重新开始，不过可以从上次的最大值开始重置。

```
    $db2 "alter table customer orders t alter column ORDER ID set generated
always as identity (start with 1, increment by 1)"
    $db2 load query table customer orders t
    Tablestate:
      Reorg Pending
    $db2 reorg table  customer orders t
    $db2 load query table customer orders t
    Tablestate:
      Normal
    $db2 "alter table customer orders t alter column order id restart with
21474836471"
```

对于序列，只能删除，然后重建后，再进行开始值的重置。

第五种办法：如果该表存放的数据是临时数据，可以将该表重命名，然后重建。

自增列：

```
rename table xxx.xxx to yyy
```

利用 db2look 导出的表定义重建表。

序列：

```
drop sequence xxx
```

利用 db2look 导出的序列定义重建序列。

3. 易失表的设计原则

所谓易失表，是指基数在运行时变化很大的表。这种表往往会造成执行计划的不正确，进而引发数据库的性能问题。比如短信队列表，在有待发送短信的时候该表是有数据的，如果所有短信都发送完成，这张表就变成了空表。如果恰巧在这个时候进行了自动的统计分析，那么这张表的 CARD 值就变为 0，后续执行计划就有可能发生问题。这种类型的表上线后运维人员很难发现，但是却像一个不定时炸弹，埋在系统内部，随时可能触发，因此需要在代码设计阶段就明确出来。

解决这种表有两种方法：一种是批量插入数据后就马上由程序自动执行一次统计分析，另一种是打开表的 VOLATILE 属性。

第一种方法在《循序渐进 DB2(第 3 版)》这本书里做过详细说明，这里不再过多叙述，这里重点说一下第二种方法。使用 ALTER TABLE 打开表的 VOLATILE 属性后，相当于将该表声明为“易失”。对于这样的表，优化器在下列情况下将执行索引扫描代替表扫描，而不考虑统计信息：

- 所引用的所有列都是索引的组成部分。
- 该索引可以在索引扫描期间应用谓词。

也就是说，对于这种表，DB2 会尽量根据查询条件使用索引扫描，而不会采用表扫描的方式。声明易失表的语法如下：

```
ALTER TABLE <TABNAME> VOLATILE
```

4.3.2　性能相关的设计原则

1. 提高插入的性能

一条数据在写入一张表的过程中，要经历写入索引对象和写入数据对象两个过程，所以提高这两个过程都可以提高插入的性能。

要提高插入索引的性能，要注意以下几点：

- 精简索引数量，删除不必要索引，可以提高插入的性能。
- 优化索引结构，通常情况下，将相异值多的列放在复合索引的所有列的前面；特例情况是，对于按照特定维度批量导入全新数据的情况下，将此维度放在索引的第一列。比如按日新增每天的交易数据，将日字段放在组合索引的第一列。
- 使用随机索引，打散热点的索引数据，减小 latch 竞争。

要提高插入数据的性能，最有效的方法是使用表追加模式。在没有开启表追加模式之前，DB2 引擎在插入表的数据之前，要先在每个页里选择空闲空间(FSCR)，默认情况下最多找 5 次。如果找到空闲空间，DB2 引擎就会把数据插入到空闲空间里；如果 5 次后都没

有找到，那么 DB2 引擎会将数据插入到表的尾部。这种方式在通常情况下都没有问题，而且可以重复利用空间，防止表的过快增长，但是在大并发下就会有很大的性能问题。原因是每张表都会有几个公共页存放 FSCR 的标记信息，DB2 引擎在查找和修改 FSCR 标记信息的过程中，需要获取该公共页面的 latch，在大并发下，就会产生很严重的 latch 竞争问题，严重影响插入性能。根据我们的观察，并发越大，对性能的影响也越大，比如单并发下通常一个插入只有零点几毫秒，但是增大到 300 并发的话，插入就可能要到几十毫秒。打开表追加模式后，DB2 引擎就不会再查找 FSCR，而是直接将数据插入到表的末尾。这种方式下，大并发的插入性能几乎不会受到太大的影响，但是带来的一个弊端是插入的表会无限扩大，因此就算删除了之前的数据但是没有重组，插入也不会再复用空闲的页面。表的追加模式可以在线打开和关闭，语法如下：

```
alter table <tabname> append on;
alter table <tabname> append off;
```

要想释放空间，在删除数据后需要对表做重组操作，或者利用分区特性，卸载历史分区才行。

2. 提高并行性

在高并发的 OLTP 环境下，比较容易出现下面两种并发问题：

1) 锁的问题

2) latch 的问题

在这两种情况下，最容易发现的一个问题就是锁的问题，主要现象是存在大量的锁等待、锁超时现象，严重的话还会发生死锁现象等。latch 问题比较难以发现，需要有经验的 DBA 才能分析出来，同时可能还需要配合 stack 和 trace 来分析原因。因此，OLTP 系统在上线前一定要进行严格的并发测试，测试过程中 DBA 要介入，观察数据库是否有这些异常。隔离级别和锁相关的介绍都在《DB2 数据库性能调整和优化(第 3 版)》里有详细介绍，因此在此只做一些有关 OLTP 系统的经验总结。

1) DB2 应用的开发人员必须考虑隔离级别问题，DB2 中默认的隔离级别是 CS。对于大多数场景，使用默认的隔离级别即可。不建议修改 DB2 默认的隔离级别，个别场景如果要修改的话，可以在语句级别进行修改。

2) 如果要求数据高度稳定，建议使用 RS 级别；如果不要求高度稳定，对于只读操作使用 UR 隔离级别，其他操作使用 CS 隔离级别。

3) 数据库的 LOCKLIST 参数要足够大，但是不要超过 2GB。过小的 LOCKLIST 会造成锁升级，增加锁等待的概率，过大的 LOCKLIST 会严重影响 DB2 引擎获取锁的性能。

4) CUR_COMMIT 选项如果没有特殊要求，建议打开。打开该选项后，读操作已经不

需要再等待其他的修改操作落实后再返回值，而是直接返回该行未变更前的值(也就是当前已落实的结果值，忽略任何可能发生的未落实操作)。这样可以极大避免死锁和锁等待的现象。

5) 在未开启 CUR_COMMIT 的情况下，如果交易系统中的典型业务场景是先查询、再更新，而且此程序被频繁地并发执行，在并发量高的情况下有可能会出现大量的死锁。因此针对此类问题，应用程序在编写 select 命令的时候必须使用 for update 选项，避免死锁的发生。在使用 for update 选项时需要详细评估，此选项会同时带来并发性下降的问题。

6) DB2_SKIPINSERTED 和 DB2_EVALUNCOMMITTED 要打开。这两个参数只是优化了 DB2 引擎读取数据的顺序问题，但是却可以极大避免死锁和锁等待的现象，并且没有什么副作用。

7) 如果同一条 SQL 语句会被同时执行，那么需要将 DB2_APM_PERFORMANCE 设置为 16，这样能避免该语句在 package cache 里的 latch 竞争问题。

4.4　代码开发的基本原则

OLTP 系统在代码的开发上也要特别注意，笔者就经历过很多上线后出现的问题，这些问题都跟开发过程不规范有关。比如有个系统上线后 CPU 使用率很高，通过分析 SQL 才发现开发人员使用了 INTEGER(TRANS_DATE)/100=200802 作为查询条件，无法命中索引。因此，提前向开发人员明确数据库的开发规范也是一项非常重要的工作。

4.4.1　命名规范

必须定义规则来统一命名，比如所有 DB2 实例、数据库对象、用户名、密码、组、文件和路径的命名都必须遵循统一的规则。

1) 所有对象的命名只能使用字母 A 到 Z、a 到 z、0 至 9 以及_ (下画线)。

2) 任何对象的命名均不能使用 DB2 中的保留字。

3) 所有数据库对象的命名必须大写。

4) 实例名长度不能超过 8 位。

5) 数据库名长度不能超过 8 位。

其他数据库对象，比如表空间、容器、模式、表、索引、字段、视图、序列、存储过程、函数、触发器、主外键、包等都需要规范命名，比如视图需要以 V_作为前缀，索引需要以 IDX_作为前缀，同时索引名称里要包含表名和列名等。

4.4.2 书写规范

必须让开发人员遵循统一的书写规范，否则开发出来的代码惨不忍睹，并且会对运维工作造成很大的困难。

好的书写规范如图 4-4 所示：

```
SELECT  FIRSTNME
        ,LASTNAME
        ,EMPNO                AS    EMPLOYEE_ID
        ,SALARY               AS    EMPOYEE_SALARY
FROM    EMPLOYEE
WHERE   SALARY > 1500.00
```

图 4-4 SQL 书写规范

1) SELECT 语句的选择字段按每行一个字段的方式编写。

2) SELECT 与首个选择的字段间有两个字符的缩进量。

3) 其他字段前跟“,”，然后是字段名；与首字段对齐。

4) 字段分隔符“,”紧跟在第二个字段的前面。

5) 字段别名“AS”语句与相应字段在同一行。

6) 多个字段的“AS”尽量对齐在同一列。

7) 在 SELECT、INSERT 和 UPDATE 等语句中，要使用明确的字段名，从而保持良好的可移植性，不允许使用“*”来代替列名。

8) 在整个 SQL 语句中，一律使用大写。

9) SELECT 语句中用到的 FROM、WHERE、GROUP BY、HAVING、ORDER BY、JOIN、UNION 等要另起一行。

4.4.3 开发规范

1. 基本规范

在基于数据库的程序开发过程中，建议开发人员遵循下面几条基本准则：

1) 程序在开发过程中应实现异常处理机制，即在数据库异常的情况下，能够通过既定的技术方案将业务数据恢复至一致状态。

2) 完全按照设计文档进行开发。

3) 程序模块内聚度要高、外联度要低。

4) 要有正确、全面的故障对策。

5) 程序编写结构合理，条理清晰。

6) 程序要按照统一的命名规则进行命名。

7) 要充分考虑程序的运行效率，包括程序的执行效率和数据库的查询、存储效率。在保证应用的同时要尽量使用效率高的处理方法。

8) 程序注释要详细、正确、规范。

9) 除非应用特别需要控制 commit 和 rollback 的提交时机，否则必须在存储过程结束时执行显式的 commit 或 rollback 操作。

10) 程序处理尽量支持 7×24 小时；对于中断，应用程序提供安全、简单的断点处理。

11) 提供标准、简单的应用输出，为应用维护人员提供明确的进度显示、错误描述和运行结果；为业务人员提供明确、直观的报表、凭证输出。

12) 要尽量使用连接池机制，连接使用完后要立刻归还给连接池。

13) 所有 SQL 操作都需要提交，如果没有配置自动提交机制，必须手工进行提交操作。

2. O/R Mapping 工具的使用规范

O/R Mapping 工具封装了 SQL 的实现，使得程序员可以使用面向对象语言来操作数据，提高了开发的便利程度。但是如果使用不慎，很有可能造成数据库性能的降低，所以在挑选和使用 O/R Mapping 工具的时候一定要慎重。上线后由于 O/R Mapping 工具造成的性能问题也比比皆是，比如生成了非常复杂的 SQL 语句，造成执行计划的混乱，又比如数据库升级后老的 O/R Mapping 工具出现了功能问题等，因此在挑选和使用 O/R Mapping 工具的时候有下面几点注意事项：

- 如果该工具生成的 SQL 语句性能不佳，必须支持自定义调优后的 SQL 语句交由该 O/R Mapping 工具执行。
- 对对象进行查询时，要正确设置查询的对象级别，确保只有需要的数据被查询出来，切莫级联将对象的子对象全部查出，对数据库造成不可评估的负载。
- 是否可以用于商业用途，如果出问题是否有支持等，是否有能力修改源代码。

3. 对于查询语句的使用规范

1) 限制 select 语句的准则

- 必须明确列出所选择的列的名称，不允许使用星号(*)代替。
- 尽量避免对 where 子句中的关联列使用类型转换和函数运算。
- 尽量避免使用空操作表达式(coalesce)，DB2 优化器无法详细分析该谓词。
- 尽量避免使用外连接，除非应用要求必须使用。
- 合理使用谓词关联，杜绝多表关联时出现笛卡尔积运算以及冗余谓词。若 from 子句中包含 n 个表，where 子句中至少存在 n-1 个关联条件。

- 为了减少发生排序操作的可能性，避免出现不必要的 DISTINCT、ORDER BY、GROUP BY、UNION 之类的子句。在不影响结果集的情况下，推荐使用 UNION ALL 和 EXCEPT ALL。
- 如果查询需要的行数远远小于结果集返回的行数(比如数据采样)，必须使用 fetch first …only 或 optimize for n 子句。
- 使用游标时明确表达语句的目的，对于只读操作，使用 for read only 选项；对于更新操作，使用 for update 选项。

2) 类型转换的处理

当查询语句中类型转换无法避免时，可以借助用户临时表优化查询。将转换后的数据插入到临时表中，在临时表上创建适当的索引，再进行关联。

对局部谓词使用表达式的情况，应该使用翻转形式进行优化。比如：

```
    INTEGER(TRANS_DATE)/100 = 200802
```

应该重写为：

```
    TRANS_DATE BETWEEN 20080201 AND 20080229
```

若应用程序能够强制执行数据一致性和完整性的保证，建议使用参考约束提高查询的性能，示例如下：

```
    create table sample table(
     name char(10),
     gender char(1) not null constraint GENDEROK check (gender in ('M','F'))
not enforced enable query optimization, age int)
```

在交易型系统中，即使频繁执行简单的 SQL 语句，也会由于语句编译工作而导致 CPU 使用率过高。虽然动态语句的执行计划可以在高速包缓存中查找到，但是谓词中使用的字面值不同会直接导致已有执行计划不匹配，从而再次进行编译工作。优化此类问题的方法是使用参数标记，比如：

```
    SELECT AGE FROM EMPLOYEE WHERE EMP ID = 26790
    SELECT AGE FROM EMPLOYEE WHERE EMP_ID = 77543
```

被看成两条不同的语句进行重复编译，应该改为：

```
    SELECT AGE FROM EMPLOYEE WHERE EMP_ID = ?
```

3) 进行复杂查询的原则

- 限制表连接操作所涉及的表的个数

对于数据库的连接操作，我们可以简单地将其想象为一个循环匹配的过程。每一次匹配相当于一次循环，每一个连接相当于一层循环，则 N 个表的连接操作就相当于一个 N-1 层的循环嵌套。

一般情况下在数据库的查询中涉及的数据表越多，其查询的执行计划就越复杂，大并发下执行的效率就越低，就越容易消耗 CPU 资源，为此我们需要尽可能限制参与连接的表的数量。建议 OLTP 系统中参与连接的表的个数不要超过 8 个。

- 限制嵌套查询的层数

应用中影响数据查询效率的因素除了参与查询连接的表的个数以外，还有查询的嵌套层数。对于非关联查询，嵌套的子查询相当于使查询语句的复杂度在算术级数的基础上增长；而对于关联查询而言，嵌套的子查询相当于使查询语句的复杂度在几何级数的基础上增长。

因此，降低查询的嵌套层数有助于提高查询语句的效率。

对嵌套查询层数的限制要求：如果查询语句拥有过多的嵌套层数，那么会使该查询语句的复杂度高速增加，应该在数据库设计阶段就避免这种情况出现，不应多于 5 层。

- 灵活应用中间表或临时表

在对涉及较多表的查询和嵌套层数较多的复杂查询的优化过程中，使用中间表或临时表是优化、简化复杂查询的一个重要方法。

4. DML 语句的调整原则

DML 语句包括 insert、update、delete 和 merge。在使用 DML 语句的时候，我们也会遇到性能低下的情况，可以参考以下内容来做出调整。

1) insert 操作缓慢并且占用过多的 I/O 资源。

这种情况发生在 PCTFREE 较高且行记录较大，频繁地寻找新的空闲数据块的时候。对于批量的 insert 操作，可以将表的参数设置为 APPEND ON，在插入结束后再改回 APPEND OFF 状态。

在数据对象有(多个)索引的情况下，insert 操作还需要对索引进行维护，这额外增加了数据插入的成本，所以对于含有过多索引的表的维护是比较耗费资源的。

2) update 操作缓慢。

update 操作需要获得操作对象上的独占锁，如果其他用户已经占有该对象的非兼容的锁，那么 update 操作就需要等待，通常这是非常短的时间。但是如果该用户在操作时被打断，该用户持有这个锁的时间就可能变长，造成其他用户的等待，这是一个管理上的问题。

如果 update 操作扩展了一个 varchar 列，导致发生行溢出，其更新也会变慢。

对于 update 操作，可以考虑对 update 的条件做索引，保证 update 先走索引扫描，再去更新数据。同时可以调整表的 PCTFREE 参数，防止出现行溢出。

3) delete 操作缓慢。

由于删除操作会产生大量的日志信息，因此对系统性能的影响较大。如果可能，可以使用更改状态标志以及在另外的表中插入新的记录来代替删除操作。对于清空全表的操作，可以用 load 命令来实现，在 DB2 V9.7 之后还可以使用 truncate table。例如：

```
load from /dev/null of del replace into <schema>.<table> nonrecoverable;
truncate table <schema>.<table> immediate;
```

4) 约束对 DML 语句的影响。

- 约束会对 DML 操作的性能产生影响。
- 完整性约束：时间会耗费在验证每一个数据值是否合法上。
- 主键约束：主键约束是由唯一索引来强制实施的，而且这个索引在插入和更新操作上的开销会使大容量的插入操作和更新操作运行变慢，因为每个值都必须从索引中查询一次。
- 外键约束：强制实施交互表之间的数据关系，必须访问外部的数据表以确认当前值是否合法，才能进行插入。
- 其他约束：对于其他检查，数据也需要做相应的检查。
- 触发器：触发器对 DML 的执行效率也有较大的影响，特别当触发器的类型为 for each row 的时候。

在执行大容量的插入或更新任务时，可以暂时禁用所有与所影响数据表有关的约束、触发器，装载数据，最后才重新启用约束。但是在启用约束时，需要考虑非法数据。

5) 索引的 DML 语句的影响。

在记录被插入和修改时，表上所有的参与索引都必须实时地进行更新。这通常会产生由于大量排序而增加的系统开销，严重降低系统的执行性能。因此调整原则是：

- 在大型的 DML 批操作中，在更改数据表之前，删除全部索引。在批操作之后，重新建立起索引。
- 如果索引因为不平衡而产生拆分等额外操作，那么选择重建索引操作，这样也会减少维护索引所需的时间。

6) DML 语句对日志的影响。

由于 DML 操作会占用日志空间，如果长时间不提交，会导致数据库空间日志占满，导致事务回滚；因此对于无法评估的大数据量的 DML 操作，要控制一次操作的数据量，采用分批提交的方式。另外切莫产生单事务处理时间过长的问题，对于长事务，应该设置多个保存点，分段提交。

在删除数据的时候，为了防止占用过大的日志空间，应该采用批量删除的方式。示例代码如下：创建一个存储过程，循环删除数据，每次删 5000 条，传入需要删除的日期(比如 20160730)作为参数。

```
CREATE PROCEDURE del data
(IN dt char(24))
LANGUAGE SQL
BEGIN
declare delcount int default 0;
      repeat
      delete from (SELECT 1 FROM nxz.test where created tx stamp < dt FETCH
FIRST 5000 rows only);
      get diagnostics delcount = row count;
      COMMIT;
      until delcount = 0
      END repeat;
END
```

4.5　本章小结

本章主要讲解了 OLTP 系统在设计和开发上的基本原则，比如如何选择软硬件，如何进行物理设计，如何提高高并发下的系统性能，代码开发上有哪些注意事项等。本章的很多内容在之前的章节或《循序渐进 DB2(第 3 版)》和《DB2 数据库性能调整和优化(第 3 版)》中都有介绍，但是需要大家可以将这些知识融会贯通起来，根据自身特点，灵活地运用到自己的系统中，最大限度地发挥出数据库的性能。

第 5 章

OLAP 系统设计与管理

上一章介绍了 OLTP(联机事务处理)系统，这一章将介绍 OLAP(联机分析处理)系统的设计与管理。OLAP 是针对特定问题的联机数据访问和分析，通过对信息(维数据)的多种可能的观察形式进行快速、稳定一致和交互性的存取，允许管理决策人员对数据进行深入观察。这种系统的典型特点是：总的数据量巨大，每秒处理的事务数可能比较低，但每个事务都相对更复杂，单个事务处理的数据量更大。

当数据库和数据仓库变得越来越大时，管理不断增长的存储数据成为影响 RDBMS 性能的关键因素。要应对数据库的增长，数据库系统必须具有可扩展性能，以便能够应用额外的计算资源。DB2 DPF 提供的数据分区功能支持将单个数据库扩展到多个服务器上。有了分区以后，就可以实现跨分区分布数据。

使用 DPF 之后，数据库将变得可伸缩，因为可以根据数据增长添加新机器或节点，并将数据库扩展到它们上面。这意味着每个附加机器可为数据库提供更多的 CPU、内存、磁盘。由于以上优点，DPF 是 OLAP 系统最常用的完美选择。

DB2 V10.5 引入了列组织表，为 OLAP 系统的设计提供了新的选择。列组织表不仅是数据组织方式上的变化，同时还有对内存、CPU 和 I/O 等多方面的优化，能够更好支持具有复杂查询的分析型工作负载，这些新功能统称为 BLU 加速。

本章我们将讲解 OLAP 系统的设计和管理，主要讲解如下内容：

- DB2 多分区(DPF)基本架构和相关概念
- DB2 多分区(DPF)应用场景和一种 OLAP 高性能架构设计

- 配置 DB2 多分区(DPF)环境
- 基于 DPF 架构的 OLAP 系统最佳实践
- DB2 列组织表的基本概念、应用场景
- DB2 列组织表的创建和装入数据以及访问计划

5.1 DB2 DPF 多分区基本架构和相关概念

5.1.1 DB2 DPF 基本架构

图 5-1 给我们展示了 DB2 DPF 数据库的基本架构。从图中可以看到，在 DB2 DPF 环境中，数据库在非共享的环境中被分解为独立的分区，每个分区都具有自己的资源，例如内存、CPU 和磁盘，以及自己的数据、索引、配置文件和事务日志。DB2 DPF 的这种架构称为 share-nothing 体系结构。数据库分区有时称为节点或数据库节点。

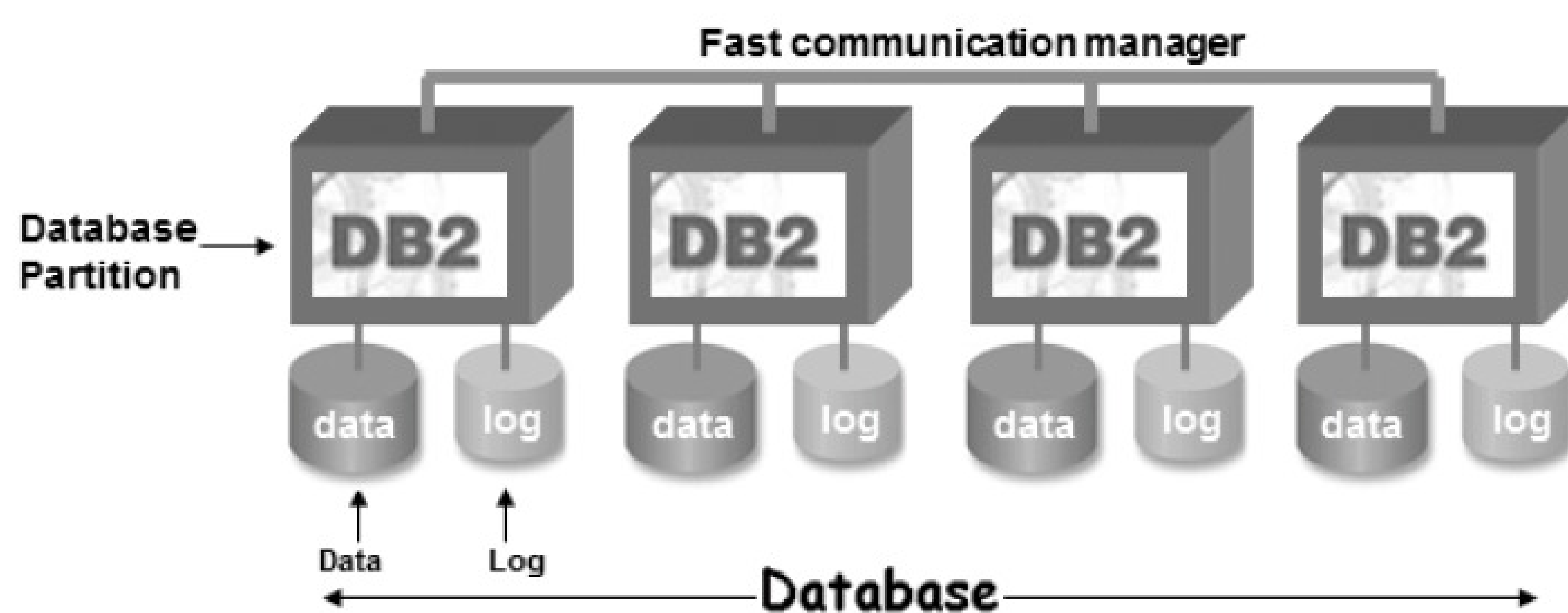


图 5-1　DB2 DPF 数据库结构

在 DB2 DPF 分区数据库环境下，数据库被分成多个分区，每个数据库分区运行在各个节点上，而且每个数据库分区具有自己的资源(Engine、LogMg、LockMg、Cache 等)。通过数据库协调节点协调所有分区进行并行处理，但是从用户和应用来看是单独的系统，看起来和普通的数据库没有区别。

5.1.2 DB2 DPF 数据的分布键以及数据倾斜问题

1. 分布键选择

在 DB2 DPF 环境中创建表之前，需要考虑下列事宜：表空间可以横跨多个数据库分区。它们所跨的数据库分区数取决于数据库分区组中的数据库分区数。

可以通过如下方法来并置表：将表置于同一个表空间中；或置于另一个表空间中，该表空间与第一个表空间一起，与同一个数据库分区组相关联。

在创建表时，指定创建的表将成为若干数据库分区的一部分。当在分区数据库环境中创建表时，有一个附加选项：分布键。分布键是作为表定义一部分的键，确定用于存储每行数据的数据库分区。

如果不显式指定分布键，会使用下列默认值：

- 如果在 CREATE TABLE 语句中指定了主键，那么主键的第一列会用作分布键。
- 对于多分区数据库分区组，如果不存在主键，那么使用非长整型字段的第一列。
- 如果没有列满足默认分布键的要求，那么会不带关键字创建该表(这只在单一数据库分区组中允许)。
- 必须小心地选择适当的分布键，因为以后再也不能更改。此外，必须将任何唯一索引(因此也是唯一键或主键)定义为分布键的超集。换言之，如果定义了分布键，那么唯一键和主键必须包括与分布键相同所有的列(它们可能有多列)。

2. 数据如何分布

DB2 DPF 使用哈希方式自动分布数据，在实际应用环境中，多分区数据库和单分区数据库相比需要注意的就是：在创建表时需要选择合适的分区键以便数据能够均匀分布到每个节点上。假设我们创建如下两张表并存储数据到表中，数据库会根据每条插入的数据中分区键的哈希值将数据分布到相应的节点上。图 5-2 描绘了这一基本过程。

```
CREATE TABLE customer (
 cust_id VARCHAR(80)
,gender  CHAR(5))
DISTRIBUTE BY HASH(cust_id);

CREATE TABLE sales (
 cust_id VARCHAR(80)
,qty     INTEGER)
DISTRIBUTE BY HASH(cust_id );
```

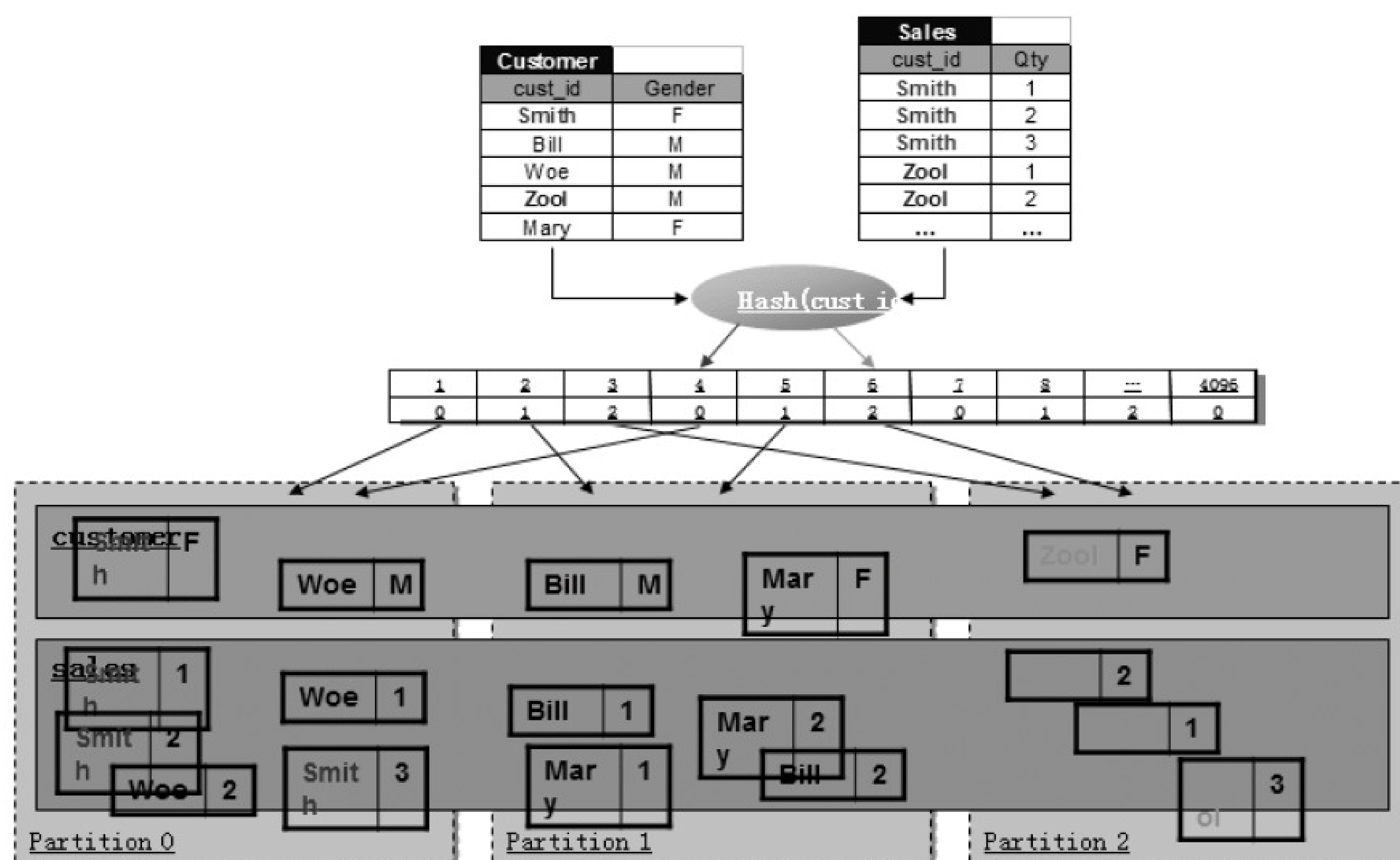


图 5-2　使用哈希方式自动分布数据

DB2 DPF 使用哈希方式自动分布数据，在日常生产中很容易出现数据倾斜到某一个节点，导致这个节点上的数据远远多于其他节点，这种情况下在应用访问时可能会导致严重的性能问题。而引起这种问题的原因就是在创建表时分区键的选择不合理。

在我们从单分区环境迁移到多分区环境时，建议尽可能选择在业务中频繁使用的且数值分布比较均匀的字段作为分区键，这样在后续的应用中将能极大地提高性能。

5.1.3　DB2 DPF 数据库的并行 I/O

DB2 DPF 数据库的并行 I/O 是采用数据库分区最主要的原因之一。将一个大的数据库分成多个小的数据库可以提高查询的性能，因为每个数据库分区都拥有自己的一小部分数据。假设您想扫描 1 亿条记录，对单一分区的数据库来讲，该扫描操作需要数据库管理器独立扫描 1 亿条记录；如果您将数据库系统做成 50 个分区，并将这 1 亿条记录平均分配到这 50 个分区上，那么每个数据库分区的数据库管理器将只扫描 200 万记录。

图 5-3 描绘了在 DPF 环境下，当应用发起查询时，语句在协调节点编译完成之后，拆分若干个子任务在每个节点上执行，最终将查询结果反馈给协调节点并反馈给应用。

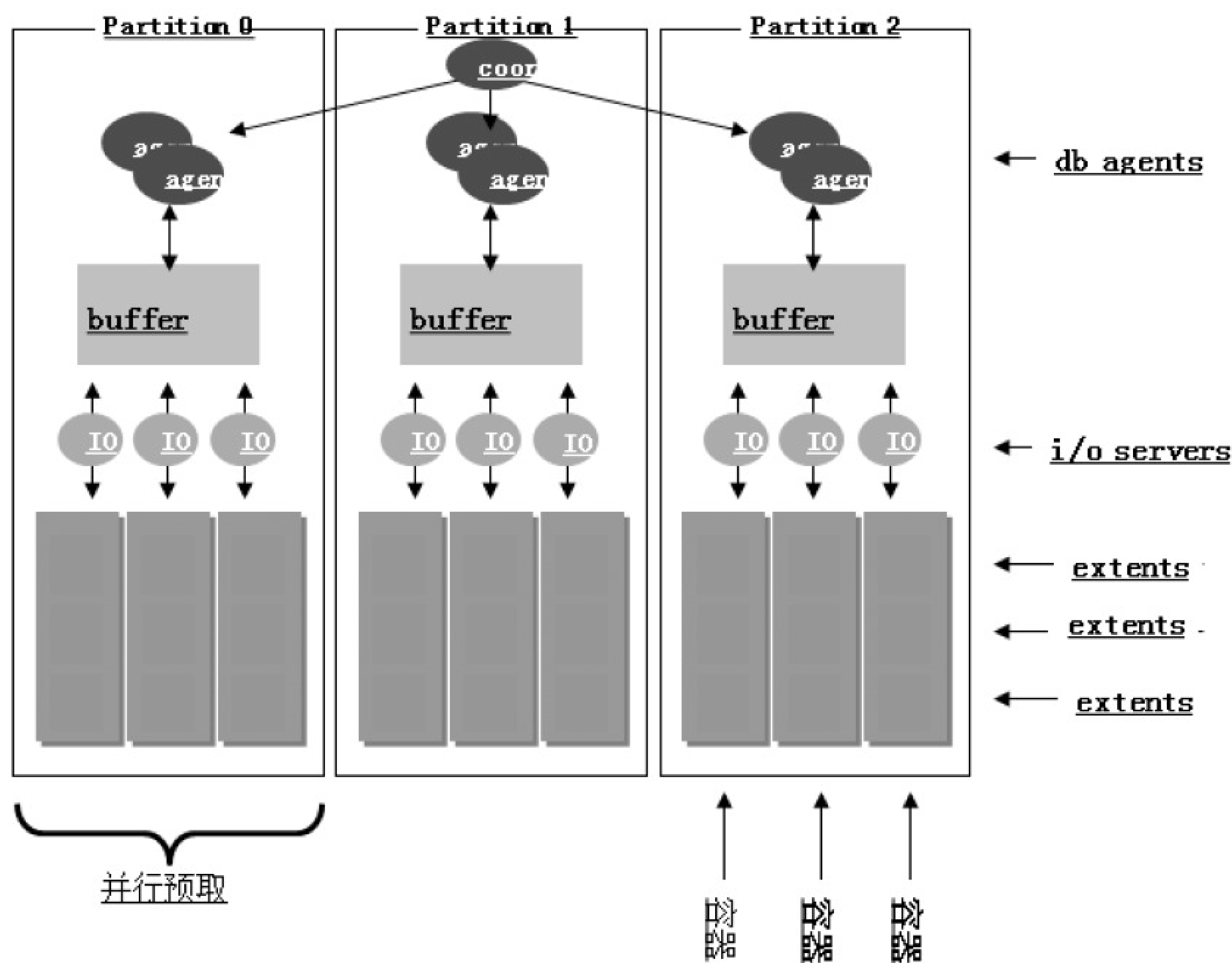


图 5-3　DB2 DPF 数据库的并行 I/O

5.1.4　DB2 DPF 数据库的扩展性

在 DB2 DPF 数据库环境中有以下两种扩展方式，从图 5-4 中可以看到这两种方式为：

- 垂直方式：单个服务器增加 CPU、内存和硬盘。
- 水平方式：向集群增加服务器节点和硬盘。

DB2 DPF 数据库环境的这种扩展方式，大大加强了整个数据库的 CPU 计算能力、硬盘的存储能力和 IO 吞吐量。

表和表空间大小限制是每个数据库分区上的所能存储的数据大小限制，因此将数据库分成 N 个分区可以将表的最大尺寸增加为单个数据库分区中表最大尺寸的 N 倍。内存也可能是个限制，因为每个数据库分区管理并拥有自己的资源，因此通过数据库分区可以克服这个限制。在业务繁忙的数据仓库系统中，往往 CPU 计算能力成为数据库的性能瓶颈，在 DPF 环境中我们可以根据需求扩展相应的资源到集群中。

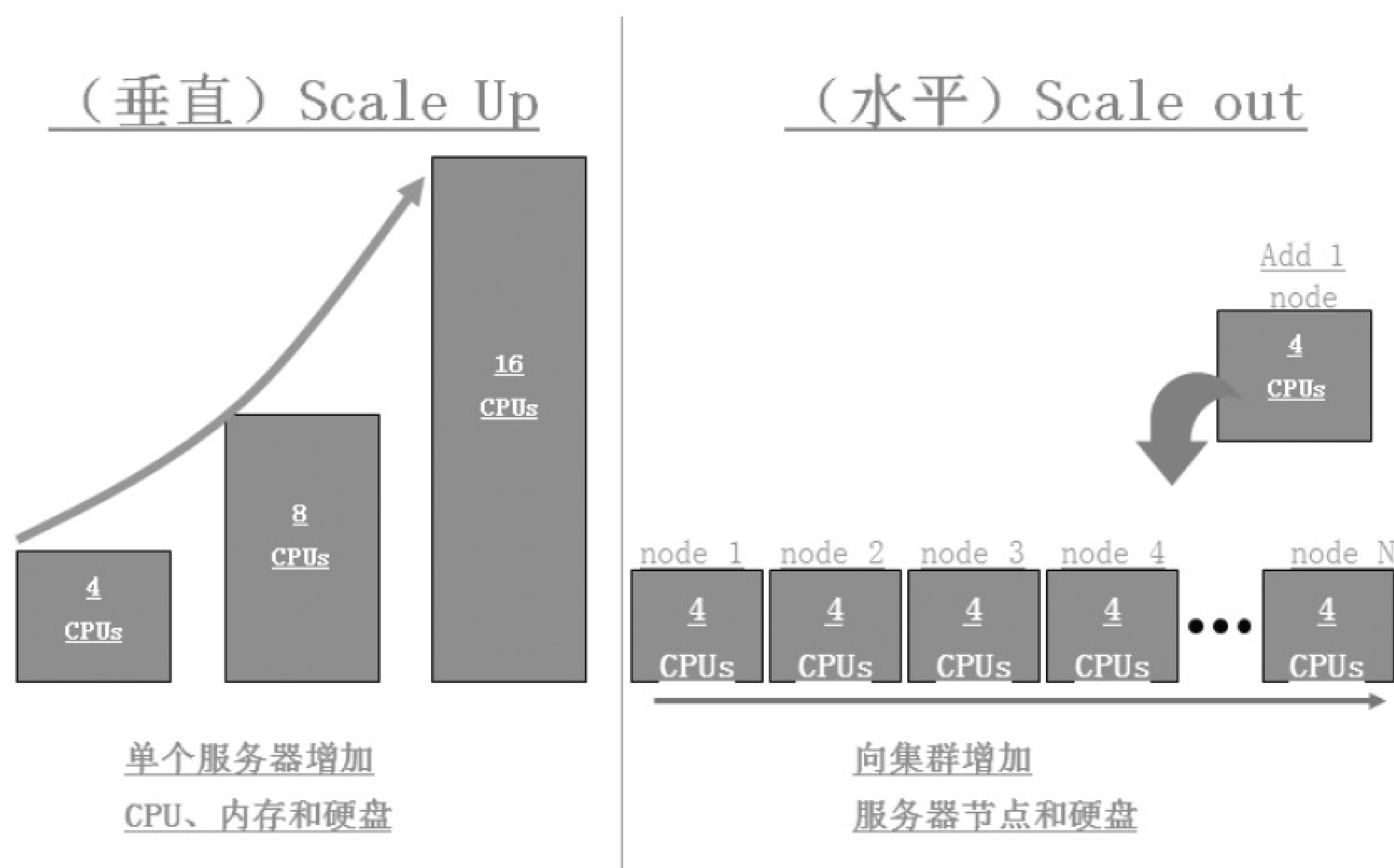


图 5-4　DB2 DPF 数据库的扩展性

DB2 DPF 的扩展方式有如下三种：

- 数据分区不发生变化，在单个服务器内增加 CPU、内存和存储，无需重分布数据。
- 数据分区不发生变化，增加服务器节点，无须重新分布数据，数据仓库管理系统需要重新启动。
- 数据分区发生变化，增加服务器节点，需要重新分布数据。
  - 支持在线的数据重新分布。
  - 仅正在重分布的那张表不可以读写，其他表都可以读写。
  - 支持在中断重分布操作后，继续重分布操作。

在对 DB2 DPF 环境进行扩展时，一定要根据当前系统的瓶颈合理地进行扩展选择，避免系统出现太大的变动，导致长时间恢复而影响业务。

5.2　DB2 DPF 多分区应用

DB2 DPF 数据库在性能上的扩展主要体现如下情况：

- 加倍的资源，加倍的数据：每个分区像以前一样处理相同的数据量，而响应时间以及吞吐量保持不变。
- 双重的资源，保持数据不变：每个分区像以前一样处理一半的数据量，而响应时间将会减半，同时吞吐量将翻一番。

● 保持资源不变，加倍数据：每个分区像以前一样处理双倍的数据量，而响应时间增加一倍，同时吞吐量将减少一半。

用户可以在逻辑服务器(在较大的 SMP 中)或物理服务器之间使用 DPF 对数据进行分区。图 5-5 中显示的就是分区数据库横跨多个物理服务器(虽然每个服务器一般都是小型 SMP 服务器)的例子。

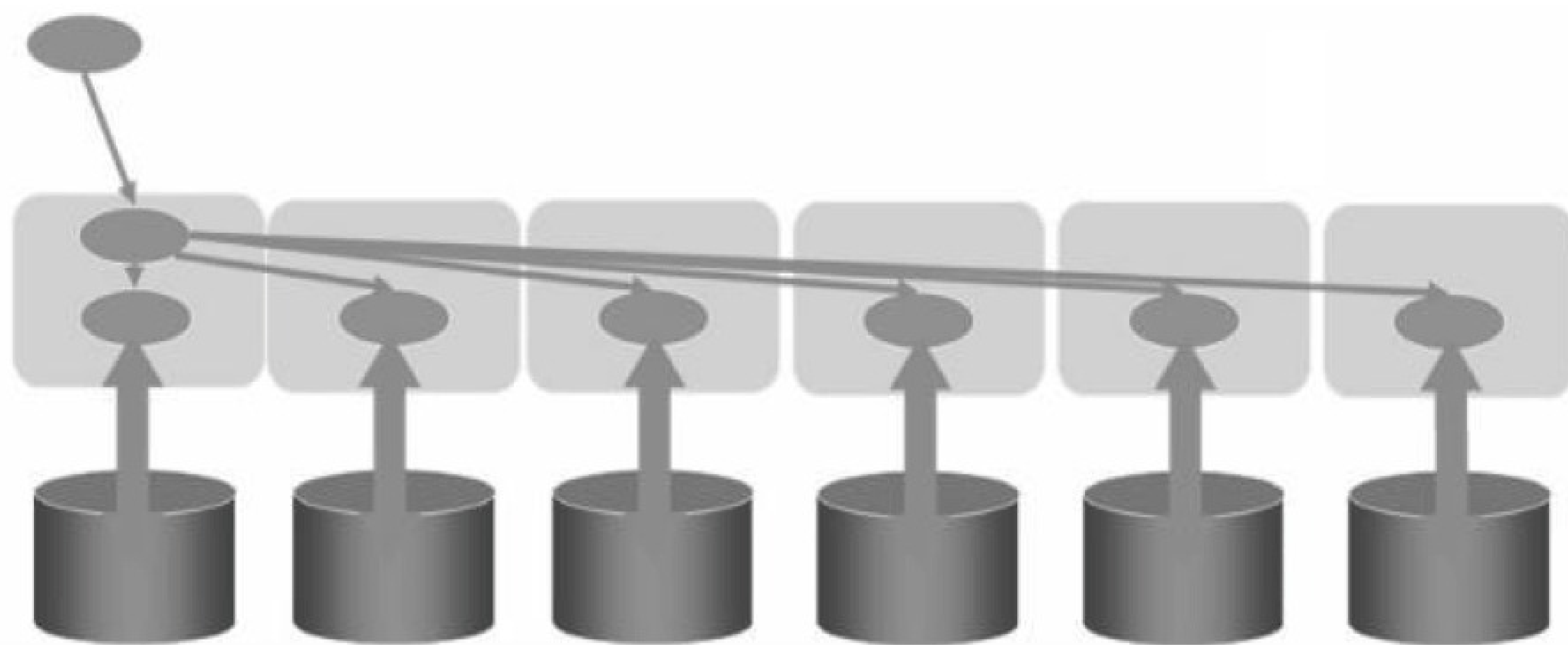


图 5-5　使用 DPF 横跨多个物理服务器的 DB2 分区

在图 5-5 中可以看到，实际上有 6 个副本，但是对于应用程序以及管理员来说都被当作单独的副本。现在试想一下在单独的 DB2 上扫描 600 000 行以及在每一个服务器上拥有自己的数据和资源而仅仅扫描 100 000 行时的情形，这就是 DPF 的威力。

DB2 中的并行是自动化的并且扩展到了 DPF 的哈希分割算法中。如果用户只选择很小的一部分记录(分区键值为 X)，DB2 将只发送该查询到包含这些数据的节点。但是如果用户要扫描大量的数据(比如图 5-5 中涉及的环境，尤其是数据仓库环境中)，那么 DB2 将会把该查询发送到集群中的所有分区，并且自动并行数据访问操作以获取更多资源(如内存、CPU 以及 I/O)来使该项工作执行得更快。

DPF 传送的不仅仅是查询性能更快的维护操作，对于每一个服务器都有明显的资源节省，因为每个服务器拥有数据的 1/n，而且一般只需要很少的资源。接下来对比图 5-6 中使用分区数据库(左边)和不使用分区数据库(右边)的情况。

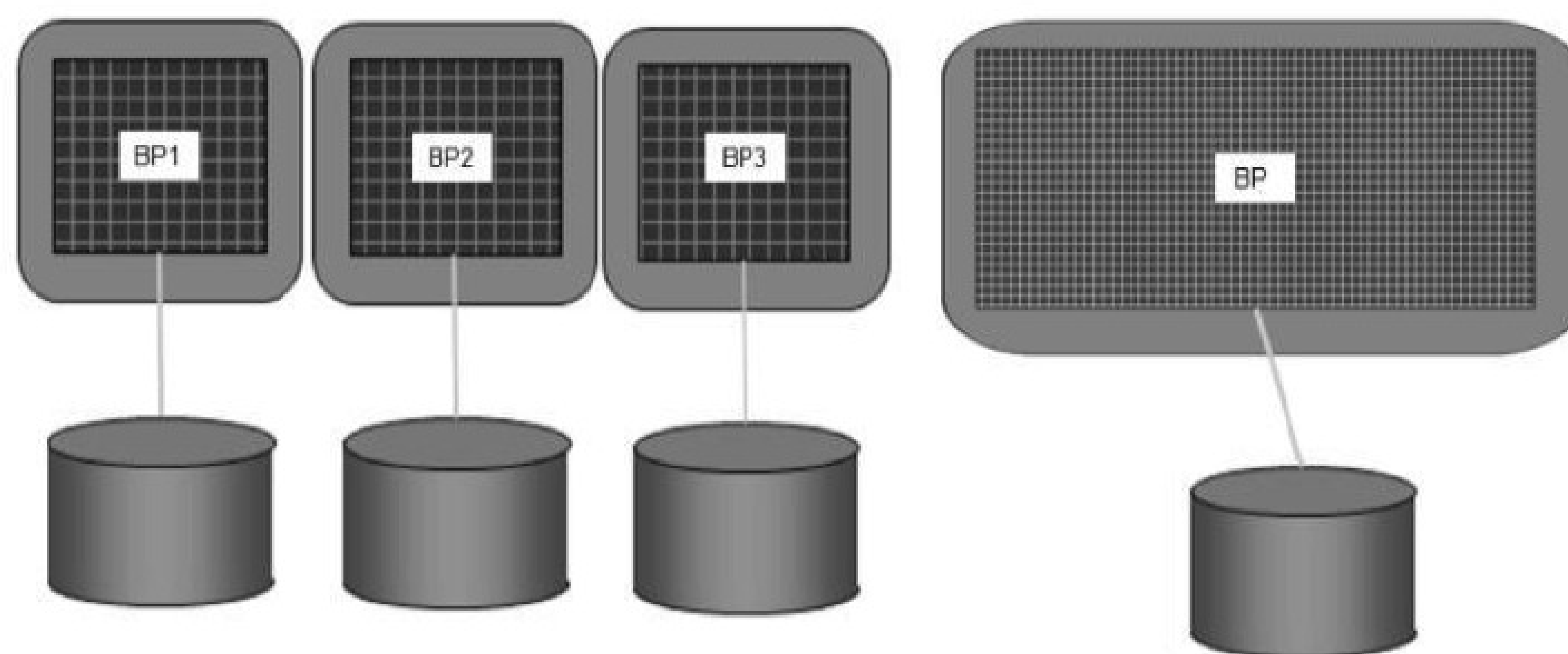


图 5-6　利用数据库分区功能节省资源

在图 5-6 中可以看到：左边包含分区数据库的服务器只需要很少的内存，并且每个服务器只需要响应 1/n 的数据，而右边的非分区数据库需要为缓冲池分配大量的内存以容纳数据。

最后，DB2 可以将内分区并行(在单独服务器中并行地运行 SQL 语句)能力与外分区并行(DPF 功能)能力混合起来使用，如图 5-7 所示。

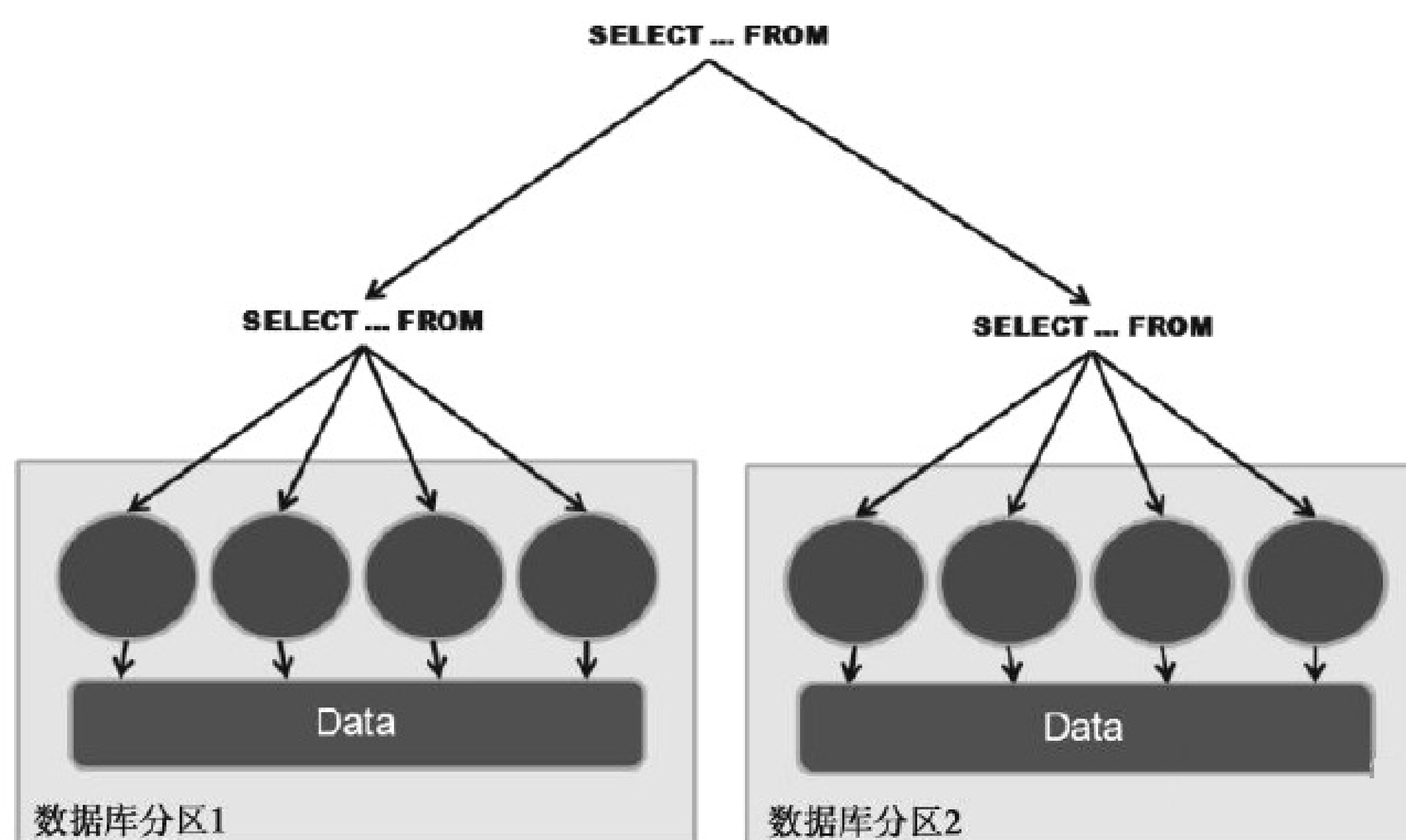


图 5-7　两种并行方式

用户应该在以下情况中考虑使用 DPF 功能包：

- 数据库应用程序操作的速度是用户业务操作的关键。一些类似重组、备份等操作可以使用 DPF 并行执行，这样可以执行得更快。例如，如果图 5-5 中的 6 个服务器都在运行备份程序，那么每台服务器只需要备份数据的 1/6，该操作将会是非分区数据库操作时间的 1/6。

- 当长期提取、转换以及加载处理需要压缩批处理窗口时。例如，数据服务器加载作业也可以并行执行，如果需要加载 600GB 的数据库，那么每一个服务器仅仅需要加载 100GB 的数据库。
- 为数据仓库进行并行 SQL 处理的滚动窗口对数据更新的需求以及额外的日志空间的需要。DPF 不仅会为用户带来更多处理 SQL 的能力，而且还会带来更多的资源(日志空间、内存等)。
- 在数据库包含超过 400GB 的原始数据时应当考虑使用 DPF。其他使用 DPF 的指标是表中的总行数以及扫描性能是否是工作负荷性能的关键因素。
- 用户环境中充满包含大型聚合、多表连接等的复杂查询。在拥有多个服务器的环境中执行同一 SQL 一般比在单独服务器上的同样操作返回结果要快一些。
- 服务器上有充足可用的内存。即使 DB2 拥有 64 位的内存来支持非分区数据库，但是已经证明多分区比单独的 SMP 并行能提供更多的线性可扩展性和更有效的内存使用。

5.3　OLAP 高性能设计：DPF + TP + MDC

在本书的第 2 章“DB2 表的高级特性”中，介绍了 DPF(数据库分区)、TP(表分区)和 MDC(多维集群)三个特性之间既独立又互补的关系。在本章中特别指出，DPF + TP + MDC 是非常常用的、典型的 OLAP 高性能设计。在 IBM 出版的 *Database Partitioning, Table Partitioning, and MDC for DB2 9* 白皮书中，对这种设计进行了详尽描述。

在单机环境中，如果一个表中有大量的数据，从这个表查询特定的数据时，如图 5-8 所示，就像“大海捞针”一样，需要较高的成本和较长的响应时间。

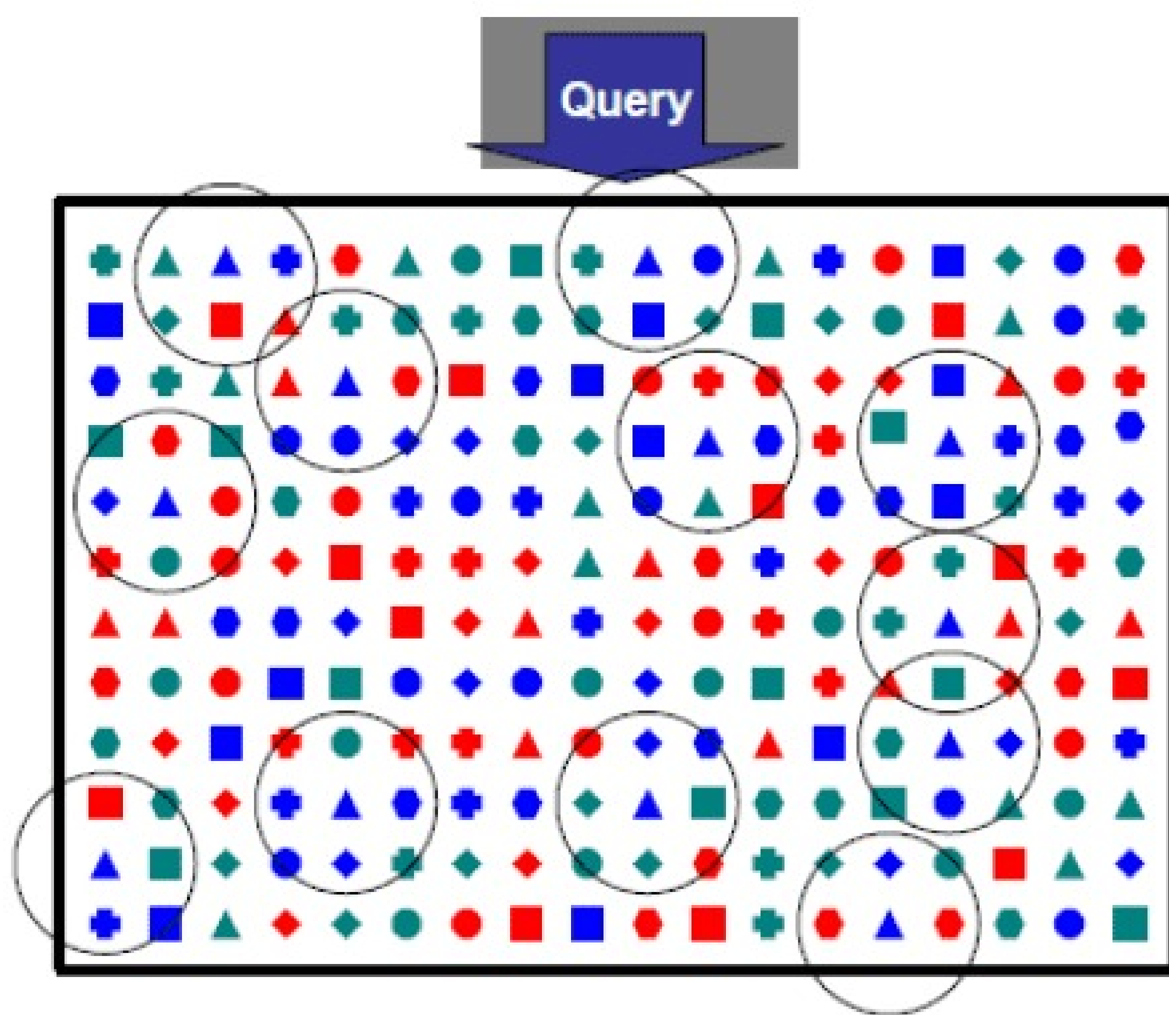


图 5-8　单机环境下从单表中查询数据

采用了 DPF 特性之后，一个大表的数据通过哈希算法被分散到 N 个数据库分区中。当从这个表查询特定的数据时，N 个数据库分区可以并行地进行查询，如图 5-9 所示，可以把响应时间降低到接近于原来的 1/N。如果要做比喻，这相当于多艘船只分别前往五大洋去搜索散落的宝藏。

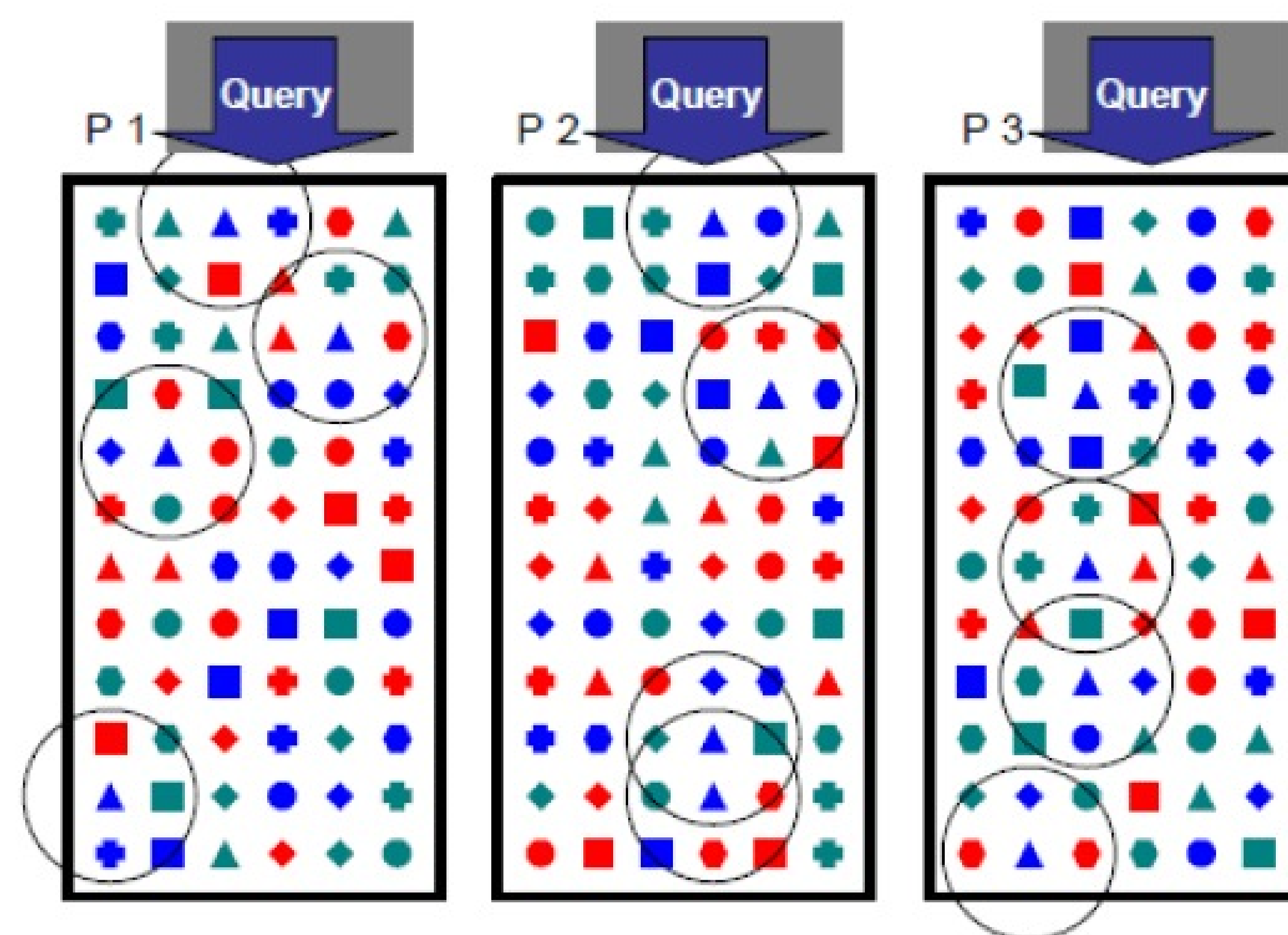


图 5-9　DPF 并行查询数据

在 DPF 的基础上再采用表分区特性之后，每个数据库分区上的数据按照指定的属性再次进行了分区。当从这个表查询特定的数据时，每个数据库分区可以并行地并且只需要在特定的表分区中进行查询，如图 5-10 所示，查询性能可以得到大幅度提升。这相当于有了粗略的“藏宝图”，多艘船只只需要前往五大洋特定的海域，去寻找散落在这片海域的宝藏。

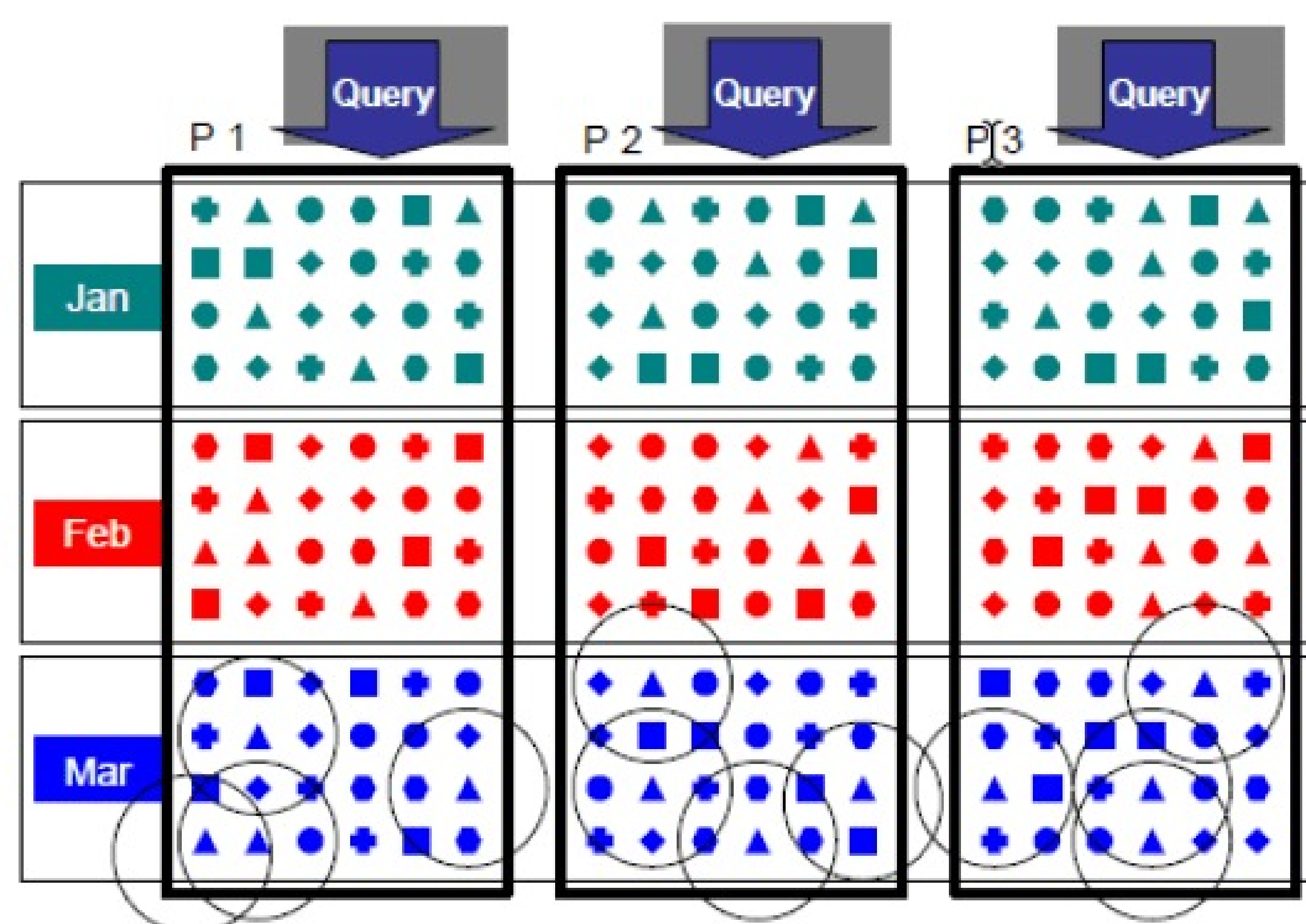


图 5-10　在 DPF + TP 中查询数据

在 DPF + TP 的基础上再采用 MDC 特性之后，在每个数据库分区中，并且在每一个表分区中，数据进一步按照特定的属性(多个列)进行了聚集。当从这个表查询特定的数据时，

每个数据库分区可以并行地并且只需要在特定的表分区中，对物理上聚集的一小部分数据进行查询，如图 5-11 所示，查询性能可以得到非常大的提升。继续拿大海寻宝做比喻，这就相当于有了详尽的“藏宝图”，多艘船只分别直奔五大洋中特定海域的特定小岛，只需要搜索几个山洞，就可以找到成箱成箱装好了的宝藏。

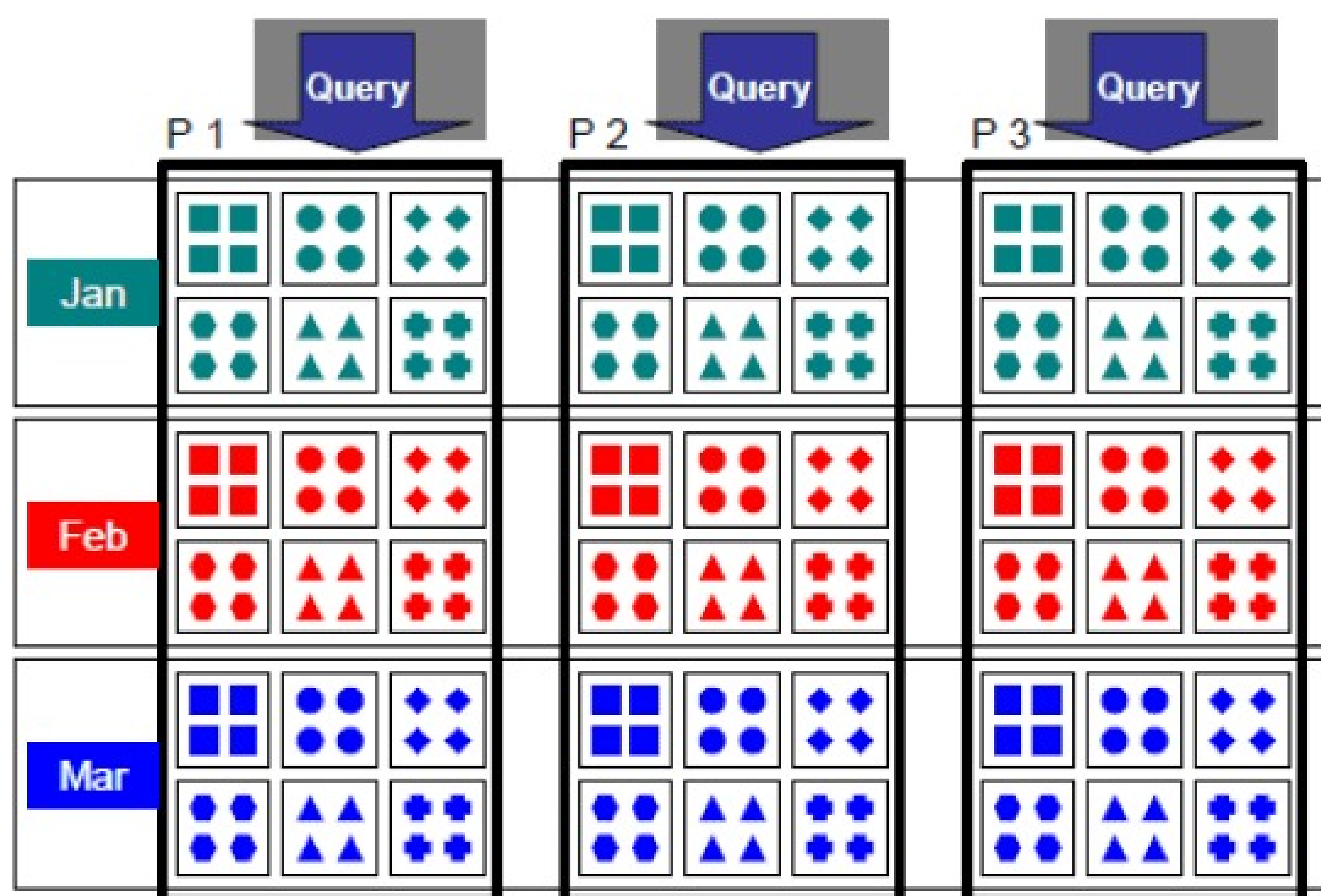


图 5-11　在 DPF + TP + MDC 中查询数据

5.4　配置 DB2 DPF 多分区环境

5.4.1　DB2 DPF 安装准备

1. 共享文件系统准备

在搭建 DB2 DPF 分区数据库之前，需要在数据库将要驻留的服务器上创建共享文件系统，在 DB2 DPF 分区数据库环境下此共享文件系统用于存储实例配置文件，以便每个节点都可以访问这些文件，使数据库能够按照 db2nodes.cfg 文件正常在各自节点启动。

目前这种共享问题系统通常使用 NFS 或 GPFS 共享文件系统，我们在这里推荐大家使用 GPFS 文件系统，具体的搭建方法建议参考 IBM 官方文档，这里不做详细配置描述。

2. 规划文件系统

在 DB2 DPF 环境搭建时另一个重要的规划便是文件系统的规划，因为 DPF 环境中是 share-nothing 结构，所以每个节点都有自己独立的数据存放路径，这些包括：数据库配置存储路径、数据存放路径、日志存放路径等，例如图 5-12 所示的文件系统规划。

DB2：db2dw			INSTANCE HOME: /db2home/db2dw/sqllib					
DBNANE	NODE	VG NAME	VG SIZE_G	lv_NAME	LV_SIZE_G	TYPE	FILESYSTEM	DESCRIPTION
		dwvgp0		lv_dwdbp0	20G	jfs2	/db2/db2dw/NODE0000	Database home director
				lv_dwdatp0	20G	jfs2	/db2/db2dw/db2datp0	Tablespace directory
				lv_dwlogp0	20G	jfs2	/db2/db2dw/db2logp0	Active logs directory
				lv_dwarcp0	20G	jfs2	/db2/db2dw/db2arcp0	Archive log directory
		dwvgp1		lv_dwdbp1	20G	jfs2	/db2/db2dw/NODE0001	Database home director
				lv_dwdatp1	200G	jfs2	/db2/db2dw/db2datp1	Tablespace directory
				lv_dwlogp1	50G	jfs2	/db2/db2dw/db2logp1	Active logs directory
				lv_dwarcp1	50G	jfs2	/db2/db2dw/db2arcp1	Archive log directory
		dwvgp2		lv_dwdbp2	20G	jfs2	/db2/db2dw/NODE0002	Database home directory
				lv_dwdatp2	200G	jfs2	/db2/db2dw/db2datp2	Tablespace directory
				lv_dwlogp2	50G	jfs2	/db2/db2dw/db2logp2	Active logs directory
				lv_dwarcp2	50G	jfs2	/db2/db2dw/db2arcp2	Archive log directory
		dwvgp3		lv_dwdbp3	20G	jfs2	/db2/db2dw/NODE0003	Database home directo
				lv_dwdatp3	200G	jfs2	/db2/db2dw/db2datp3	Tablespace directory
				lv_dwlogp3	50G	jfs2	/db2/db2dw/db2logp3	Active logs directory
	DWDB01			lv_dwarcp3	50G	jfs2	/db2/db2dw/db2arcp3	Archive log directory
		dwvgp4		lv_dwdbp4	20G	jfs2	/db2/db2dw/NODE0004	Database home directo
				lv_dwdatp4	200G	jfs2	/db2/db2dw/db2datp4	Tablespace directory
				lv_dwlogp4	50G	jfs2	/db2/db2dw/db2logp4	Active logs directory
				lv_dwarcp4	50G	jfs2	/db2/db2dw/db2arcp4	Archive log directory
		dwvgp5		lv_dwdbp5	20G	jfs2	/db2/db2dw/NODE0005	Database home directory
				lv_dwdatp5	200G	jfs2	/db2/db2dw/db2datp5	Tablespace directory
				lv_d[illegible]gp5	50G	jfs2	/db2/db2dw/db2logp5	Active logs directory

图 5-12　文件系统规划

另外，在 IBM 的官方建议中，是在存储上划分单独的 vg，单个 vg 使用自己独立的磁盘，这样在将来的数据库访问中可以避免出现热点盘。

3. 主机参数调整

在搭建 DPF 之前需要对主机参数进行适当调整，在 AIX 系统中目前都是动态调整，但是一些网络等参数建议还是适当调整，如下给出了部分操作系统参数建议，大家可以参考：

```
##修改操作系统参数(重启机器生效)
# chdev -P -l aio0 -a minservers=20 \
  -a maxservers=80
# no -p -o sb max=1310720 -o rfc1323=1 \
-o ipqmaxlen=250 -o udp sendspace=65536 \
-o udp recvspace=655360 \
-o tcp sendspace=221184 \
-o tcp recvspace=221184 \
-o tcp mssdflt=1440
vmo -p -o lru file repage=0 -o minfree=8000 -o maxfree=8512
vmo -p -o minperm%=3 -o maxperm%=10 -o maxclient%=10
# ioo -p -o j2_minPageReadAhead=32 \
```

```
-o j2 maxPageReadAhead=512 \
-o j2 nBufferPerPagerDevice=2048 \
-o lvm bufcnt=16 -o minpgahead=128 \
-o maxpgahead=512 -o pv_min_pbuf=1024
```

在 HPUX 和 Linux 系统中可以根据 db2osconf 程序给出的建议进行适当调整。更多的系统参数调整建议请咨询系统工程师。

5.4.2　DB2 DPF 环境搭建

DB2 DPF 环境搭建是比较烦琐的工作，我们需要认真细心地去逐步完成每一部分操作，如下我们以搭建包含 9 个分区的数据库为例为大家展示 DPF 环境的搭建过程。

1. 安装 DB2 ESE 软件并更新 license

在 DB2 版本选择上建议大家尽量使用较新的 DB2 V10.5 或以上版本，不仅包含以往版本的众多非常有用的特性：索引可分区、在线降低表空间高水位、新的压缩算法等，还进行了各方面的优化以及缺陷修复，提高了数据库的空间利用率、性能和稳定性。在安装完成之后，DB2 ESE 版本至少需要更新一下 ESE 和 DPF 两个 license，才可保证 DB2 DPF 的正常运行。

```
# ./db2licm -a /tmp/db2ese c.lic
# ./db2licm -a /tmp/db2dpf.lic
```

2. 创建实例用户并设置 rsh 信任

在 DB2 DPF 搭建过程中，我们需要创建用户，这里以 db2dw 为实例用户。我们使用如下命令创建用户：

- 创建用户组及用户

```
mkgroup -'A' id='1000' db2adm
mkuser id=1000 pgrp=db2adm home=/db2home/db2dw db2dw
```

注意：

更改实例用户密码时必须保持每台机器上的密码相同。实例用户目录必须存放在共享文件系统中，以保障每个节点可以正常访问。

- 在创建完实例用户之后开始配置 rsh 信任

```
##以 db2dw 用户登录
$ vi ~/.rhosts
DWDB00 db2dw
```

```
DWDB01 db2dw
DWDB02 db2dw
DWDBBK1 db2dw
#加入此 3 行并保存退出
$ chmod 644 .rhosts
#测试信任是否配置成功
DWDB00:/db2home/db2dw$rsh DWDB01 date
Sun Jan 27 14:34:52 BEIST 2013
显示信息，验证成功！
```

3. 创建实例和搭建 DB2 DPF 环境

配置/etc/services，在 DPF 配置中我们单独指定端口用于外部通信，而且每个节点都需要有专用端口，必须确保在 etc 目录的 services 文件中为 FCM 通信定义足够的端口。

向/etc/services 文件中添加：

```
db2c_db2dw     50000/tcp
DB2_db2dw      65010/tcp
DB2_db2dw_1    65011/tcp
DB2_db2dw_2    65012/tcp
DB2_db2dw_3    65013/tcp
DB2_db2dw_4    65014/tcp
DB2_db2dw_5    65015/tcp
DB2_db2dw_6    65016/tcp
DB2_db2dw_7    65017/tcp
DB2_db2dw_END  65018/tcp
```

创建数据库实例：

```
./db2icrt -p db2c_db2dw -s ese -u db2dw db2dw
```

创建实例完成之后会自动在用户目录下产生 sqllib 目录，为了配置多分区数据库，我们需要配置 db2nodes.cfg 文件。

db2nodes.cfg 文件的内容为：nodenumber、hostname、logical-port 和 netname。通常，配置 nodenumber、hostname 和 logical-port 即可。

```
#修改$home/sqllib/instance/db2nodes.cfg
0  DWDB00 0
1  DWDB01 0
2  DWDB01 1
3  DWDB01 2
4  DWDB01 3
```

```
5  DWDB02 0
6  DWDB02 1
7  DWDB02 2
8  DWDB02 3
```

修改完之后，使用 db2start 命令启动数据库，启动过程如下：

```
db2start
```

我们可以通过 ps –ef |grep db2sysc 命令来查看在本服务器驻留的节点启动进程信息：

```
DWDB02:/db2home/db2dw$ps -ef|grep db2sysc |grep -v grep
  db2dw 15597806 27000920  6  Nov 14     - 3441:01 db2sysc 6
  db2dw 17301648 11141288  6  Nov 14     - 3429:05 db2sysc 5
  db2dw 18415730 26214438  8  Nov 14     - 3479:55 db2sysc 8
  db2dw 28639362 27459678  8  Nov 14     - 3572:26 db2sysc 7
```

4. 创建数据库和配置参数

- 创建数据库

在 DB2 DPF 安装之前，我们已经对文件系统进行了规划。我们规划数据库存储配置文件的路径为/db2/db2dw/NODE000*，我们在创建数据库时只需要指定到/db2 下即可。

在确保实例用户拥有对以上文件修改权限的前提下，使用如下命令创建数据库：

```
CREATE DATABASE DW AUTOMATIC STORAGE NO ON /db2 USING CODESET UTF-8 TERRITORY CN;
```

- 实例参数调整

在 DB2 DPF 环境下，实例参数的修改和普通单分区数据库的修改方式是一致的，在任何一个节点上修改都可以完成修改目的，因为实例文件创建在共享文件系统中，每个节点都可以访问，并且实例配置文件只有一套。对于实例参数，具体修改哪些建议大家根据实际情况进行调整。

- 数据库参数调整

而修改数据库参数则必须在每个节点上进行，这是因为数据库参数是独立于其他节点的，我们可以使用如下示例命令进行修改：

```
#修改日志大小、日志缓冲池和日志数量
db2 all "db2 update db cfg for dw using LOGBUFSZ 8192 LOGFILSIZ 20480 
LOGPRIMARY 100 LOGSECOND 50"
#修改日志归档路径：
db2 update db cfg for dw DBPARTITIONNUM 0  using NEWLOGPATH
```

```
/db2/db2dw/db2logp0/dw LOGARCHMETH1 DISK:/db2/db2dw/db2arcp0/
```

在数据库修改为归档日志方式之后，需要对数据库全库做全备，这样数据库才能正常访问：

```
db2 all "db2 force application all"
db2 all "<<+0< db2 BACKUP DB DW to /dev/null"
db2_all "|<<-0< db2 BACKUP DB DW to /dev/null"
```

激活数据库，使参数生效：

```
db2_all ";db2 connect to DW"
```

在激活过程中，因为刚才已经修改了日志参数，所以需要一段时间，需要耐心等待。

5.4.3　创建表空间和缓冲池

在搭建完上述 DB2 DPF 环境之后，接下来就可以对数据库进行缓冲池和表空间的创建操作。

1. 创建缓冲池

我们这里只创建 32KB 页大小的缓冲池，命令如下：

```
CREATE BUFFERPOOL BP_32K ALL DBPARTITIONNUMS SIZE 10000 PAGESIZE 32K ;
```

注意：
指定 ALL DBPARTITIONNUMS 选项会在每个分区上创建缓冲池。

2. 创建表空间

在 DB2 DPF 中还有一个概念需要我们了解，就是分区组。可使用 CREATE DATABASE PARTITION GROUP 语句创建数据库分区组。此语句指定将用来存放表空间容器和表数据的一组数据库分区。

```
#单分区分区组(1 个节点)：用于存储单分区表
CREATE DATABASE PARTITION GROUP SDPG ON DBPARTITIONNUMS (0) ;
#多分区分区组(1-8,8 个节点)：用于存储多分区表
CREATE DATABASE PARTITION GROUP PDPG ON DBPARTITIONNUMS (1 to 8) ;
```

在创建完分区组之后，我们可以在定义的分区组上创建表空间，例如在分区组 PDPG 上创建的表空间容器就在 1-8 节点的每个节点上都存在。

如下为创建表空间示例：

```
create large tablespace TBS DW DAT
in PDPG
pagesize 32k
managed by database
using ( file '/db2/db2DW/db2datap $N /TBS DW DAT' 10G )
EXTENTSIZE 32
PREFETCHSIZE AUTOMATIC
BUFFERPOOL BP_32K;
```

这里的$N 是变量，会根据您知道分区组自动在创建表空间时生成每个节点容器的路径，而不需要我们每个都指定一次。

5.5　DB2 DPF 运维操作实践

5.5.1　DB2 DPF 分区节点的扩展和删除实践

前面描述过，DB2 DPF 的扩展方式有如下三种：

- 数据分区不发生变化，在单个服务器内增加 CPU、内存和存储，无需重分布数据。
- 数据分区不发生变化，增加服务器节点，无须重新分布数据，数据仓库管理系统需要重新启动。
- 数据分区发生变化，增加服务器节点，需要重新分布数据。
  - 支持在线的数据重新分布。
  - 仅正在重分布的那张表不可以读写，其他表都可以读写。
  - 支持在中断重分布操作后，继续重分布操作。

我们在这里按照第三种方案进行操作实践，如下为具体操作步骤：

1. 扩展数据库分区

多分区环境下可以进行硬件(主机和存储)的扩展以满足系统对硬件的需求。在本书中介绍的分区添加和删除的简单步骤，在实际环境中需要根据具体情况加以考虑。

2. 添加数据库分区

为多分区数据库添加分区的操作步骤如下：

动态增加节点：

```
    db2 "db2start DBPARTITIONNUM 9 ADD DBPARTITIONNUM HOSTNAME CASHMAPP2 PORT
9 WITHOUT TABLESPACES"
```

刷新 db2nodes.cfg 配置文件，需要重新启动数据库才可以生效：

```
db2stop force;db2start
```

添加分区到分区组 PDPG：

```
db2 "ALTER DATABASE PARTITION GROUP PDPG ADD DBPARTITIONNUM (9) WITHOUT
TABLESPACES"
```

这里需要注意的是，在添加分区到分区组时，要将所有的 system tempspace 扩展到新添加的分区上，这样才能将新的分区添加到分区组中。

```
db2 "ALTER TABLESPACE TEMPSPACE1 ADD
( '/db2/db2DW/db2DW/NODE0009/SQL00001/SQLT0001.0') on DBPARTITIONNUM (9)"
```

接下来需要为每个用户表空间添加容器到新的节点上：

```
db2 "alter tablespace TBS_DW_DAT
add ( file '/db2/db2dw/db2data9/TBS_DW_DAT ' 2G) on DBPARTITIONNUM (9)"
```

扩展完之后，对数据库数据重新进行分布操作，以便数据能够分布到新的分区上：

```
db2 "REDISTRIBUTE DATABASE PARTITION GROUP PDPG uniform"
```

3. 删除数据库分区

下面示例展示了如何从数据库分区组删除分区。

从分区组把数据分区删除：

```
db2 "ALTER DATABASE PARTITION GROUP PDPG drop DBPARTITIONNUM (9)"
```

数据会有一个重新分布的过程，较慢。

删除完之后，停止并删除目标分区：

```
db2 "db2stop DROP DBPARTITIONNUM 9"
```

5.5.2 DB2 DPF 数据均衡实践

在 DB2 DPF 上线之初，很容易出现数据倾斜的问题。在实际应用环境中，大家可以按照如下操作步骤进行数据库均衡操作。

这里我们以如下表为操作案例：

```
CREATE TABLE "FRPUSER "."T54_DM_AMT_INFO"  (
                  "ACCT_NUM" VARCHAR(50) NOT NULL ,
                  "AMT_TYPE_CD" VARCHAR(2) NOT NULL ,
```

```
                    "CURR CD" VARCHAR(5) ,
                    "DR CR FLAG" VARCHAR(2) ,
                    "AMT VAL" DECIMAL(31,9) ,
                    "LST TX DATE" DATE ,
                    "LST CUST TX DATE" DATE ,
                    "N O FLAG" VARCHAR(1) ,
                    "DW VALID FLAG" VARCHAR(1) ,
                    "ETL DT" DATE )
                   COMPRESS YES
                   DISTRIBUTE BY HASH("CURR CD ")
                     IN "TBS_FRP_MAN_DAT" INDEX IN "TBS_FRP_MAN_IDX" ;
```

可以使用如下语句查看当前每个节点的数据分布情况，数据没有均匀分布到每个节点上将会存在数据倾斜问题。

```
    select dbpartitionnum(CURR CD) as PRT NUM,count(*) as ROW NUM from
FRPUSER.T54 DM AMT INFO group by dbpartitionnum(CURR CD) order by
dbpartitionnum(CURR_CD);

    PRT NUM      ROW NUM
    ----------- -----------
              1       44367
              2       39001
              3       45721
              4      240234
              5      444567
              6       45621
              7      439234
              8      644236
```

经过和应用人员沟通，发现在日常查询中他们还会经常用到 ACCT_NUM 这个字段，我们可以使用如下语句查看这个字段的值域是否均匀：

```
    SELECT ACCT NUM, COUNT(*) AS OCCURRENCES FROM FRPUSER.T54 DM AMT INFO GROUP
BY ACCT_NUM ORDER BY OCCURRENCES DESC

    ACCT NUM                                              OCCURRENCES
    ---------------------------------------------------- -----------
    630******0000275                                                1
    630******0000306                                                1
    630******0000402                                                1
    630******0000605                                                1
    630******0000744                                                1
    …
    略
```

可以看到这个字段为账号，是均匀的，我们可以使用这个字段作为分布键。修改表定义，创建备份表：

```
CREATE TABLE "FRPUSER "."T54 DM AMT INFO BAK"  (
               "ACCT NUM" VARCHAR(50) NOT NULL ,
               "AMT TYPE CD" VARCHAR(2) NOT NULL ,
               "CURR CD" VARCHAR(5) ,
               "DR CR FLAG" VARCHAR(2) ,
               "AMT VAL" DECIMAL(31,9) ,
               "LST TX DATE" DATE ,
               "LST CUST TX DATE" DATE ,
               "N O FLAG" VARCHAR(1) ,
               "DW VALID FLAG" VARCHAR(1) ,
               "ETL DT" DATE )
              COMPRESS YES
              DISTRIBUTE BY HASH("ACCT NUM ")
                IN "TBS_FRP_MAN_DAT" INDEX IN "TBS_FRP_MAN_IDX" ;
```

接下来我们就可以将原来的数据导入新的备份表中，可以通过 load cursor 方式。这种方式不会记录日志，不影响业务：

```
DECLARE curl CURSOR  FOR SELECT * FRPUSER.T54 DM AMT INFO;
LOAD FROM curl OF cursor INSERT INTO FRPUSER.T54_DM_AMT_INFO;
```

在插入完之后，再次验证数据是否还倾斜，可以从下面输出看到数据已经均匀分布到每个节点上了。之后我们将原来的表重命名，将备份表重命名为原表名即可。

```
PRT NUM     ROW NUM
----------- -----------
          1      244495
          2      239848
          3      245771
          4      240404
          5      240087
          6      245663
          7      239938
          8      244791

  8 record(s) selected.
```

5.5.3 load copy yes 以及相应的前滚方法

因为 LOAD 操作不记录日志，所以在多分区环境下如果不采用 load copy yes 方式，在

数据库异常之后进行恢复操作，在前滚时有装载操作的表将不可用。为了保证数据的安全，我们在多分区环境下也采用 load copy yes 方式导入数据。

在多分区环境下，load copy yes 使用的存储路径必须保证每个节点可见，目前 TEST 采用的是 GPFS 文件系统，三台服务器可见(/db2/db2TEST/copy_yes)。

Load copy yes 操作举例如下：

```
    load from /db2/db2TEST/backup/temp1.del of del replace into TEST.temp1 copy
yes to /db2/db2TEST/copy_yes
```

在导入数据之后在目录下会生成类似如下的文件，在数据库恢复前滚时会使用这些备份进行前滚恢复。

```
TEST.4.db2inst1.NODE0008.CATN0000.20120507165714.001
TEST.4.db2inst1.NODE0007.CATN0000.20120507165714.001
TEST.4.db2inst1.NODE0006.CATN0000.20120507165714.001
TEST.4.db2inst1.NODE0005.CATN0000.20120507165714.001
TEST.4.db2inst1.NODE0004.CATN0000.20120507165714.001
TEST.4.db2inst1.NODE0003.CATN0000.20120507165714.001
TEST.4.db2inst1.NODE0002.CATN0000.20120507165714.001
TEST.4.db2inst1.NODE0001.CATN0000.20120507165714.001
```

对于 copy yes 方式，多分区数据库在恢复数据库完成之后，不能立即前滚。我们需要从数据库历史记录中查看是否有装载操作在备份时间戳之后。

查看是否有装载操作的命令为：

```
db2 LIST HISTORY LOAD SINCE 20120507135212 FOR TEST
```

在找到时间戳之后的表之后，在 copy yes 存储路径中找到具体的备份文件。

设置全局 load recovery 配置文件：

```
db2_all "db2set DB2LOADREC=/db2/db2TEST/load/loadrec"
```

修改配置文件，配置内容具体如下：

```
TIM 20120229095908          --->时间戳
DBP 0                       --->连接分区号
SCH DB2TEST                 --->tabschema
TAB TEMP1                   --->tabname
DAT TEST                    --->database
DB2 db2TEST                 --->instance
BUF NULL
SES NULL
```

```
    TYP L                          --->使用备份介质的连接方式
    LOC 8                          --->表所设计的分区数，如下指定每个分区的归档文件路径
    ENT
/db2/db2TEST/load/TEST.4.db2TEST.NODE0008.CATN0000.20120229095908.001
    ENT
/db2/db2TEST/load/TEST.4.db2TEST.NODE0007.CATN0000.20120229095908.001
    ENT
/db2/db2TEST/load/TEST.4.db2TEST.NODE0006.CATN0000.20120229095908.001
    ENT
/db2/db2TEST/load/TEST.4.db2TEST.NODE0005.CATN0000.20120229095908.001
    ENT
/db2/db2TEST/load/TEST.4.db2TEST.NODE0004.CATN0000.20120229095908.001
    ENT
/db2/db2TEST/load/TEST.4.db2TEST.NODE0003.CATN0000.20120229095908.001
    ENT
/db2/db2TEST/load/TEST.4.db2TEST.NODE0002.CATN0000.20120229095908.001
    ENT
/db2/db2TEST/load/TEST.4.db2TEST.NODE0001.CATN0000.20120229095908.001
```

查看当前数据库是否处于 pending 状态：

```
    db2 rollforward db TEST query status
                               Rollforward Status
     Input database alias                   = TEST
     Number of nodes have returned status    = 9
     Node number  Rollforward       Next log      Log files processed       Last
committed transaction          status      to be read
     -----------  --------------------------  ------------------------------
             0  DB  pending                S0000001.LOG                    -
2012-02-29-01.57.31.000000 UTC
             1  DB  pending                S0000005.LOG                    -
2012-02-29-01.57.39.000000 UTC
             2  DB  pending                S0000005.LOG                    -
2012-02-29-01.57.42.000000 UTC
    略。
             8  DB  pending                S0000005.LOG                    -
2012-02-29-01.58.00.000000 UTC
```

对数据库进行前滚恢复：

```
    db2 "rollforward db TEST to end of logs and complete "
                               Rollforward Status
     Input database alias                   = TEST
     Number of nodes have returned status    = 9
```

```
     Node number  Rollforward                  Next log            Log files
processed        Last committed transaction            status    to be read
     -----------  --------------------------  ------------------------------
             0  not pending
S0000001.LOG-S0000003.LOG  2012-02-29-01.59.11.000000 UTC
             1  not pending
S0000005.LOG-S0000007.LOG  2012-02-29-01.59.11.000000 UTC
             2  not pending
S0000005.LOG-S0000007.LOG  2012-02-29-01.59.11.000000 UTC
             3  not pending
S0000005.LOG-S0000007.LOG  2012-02-29-01.59.11.000000 UTC
    略。
             8  not pending
S0000005.LOG-S0000007.LOG  2012-02-29-01.59.11.000000 UTC
```

5.5.4　多分区 load 失败处理

多分区数据库环境下如果 load 时发生错误，在进行 load 异常恢复时，需要注意以下情况。

在使用传统单库方式恢复时，对应 load 部分节点失败的情况下是无法恢复表状态的，报错如下：

```
    $db2 load from /dev/null of del terminate into
MDSUSER.T41 CB FINANCIAL EVENT ALL

     Agent Type     Node     SQL Code     Result
    ------------  --------  -----------  ----------------------------------
     LOAD          002      -00027902    Init error. Table unchanged.
    ------------  --------  -----------  ----------------------------------
     LOAD          003      +00000000    Success.
    ------------  --------  -----------  ----------------------------------
     LOAD          004      +00000000    Success.
    ------------  --------  -----------  ----------------------------------
     LOAD          005      +00000000    Success.
    ------------  --------  -----------  ----------------------------------
     LOAD          006      +00000000    Success.
    ------------  --------  -----------  ----------------------------------
     LOAD          007      -00027902    Init error. Table unchanged.
    ------------  --------  -----------  ----------------------------------
     LOAD          008      -00027902    Init error. Table unchanged.
    ------------  --------  -----------  ----------------------------------
     LOAD          009      +00000000    Success.
    ------------  --------  -----------  ----------------------------------
```

```
    LOAD          010      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          011      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          012      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          013      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    RESULTS:       9 of 12 LOADs completed successfully.
  ------------  -------- -----------  -----------------------------------

  Summary of LOAD Agents:
  Number of rows read          = 0
  Number of rows skipped        = 0
  Number of rows loaded        = 0
  Number of rows rejected       = 0
  Number of rows deleted        = 0
  Number of rows committed      = 0
```

多分区环境下部分节点失败需要按照如下方式进行操作，指明需要恢复的节点：

```
    db2 "load from /dev/null of del terminate into
MDSUSER.T41 CB FINANCIAL EVENT ALL partitioned db config output nodes
(3,4,5,6,9,10,11,12,13)"
    Agent Type     Node     SQL Code     Result
  ------------  -------- -----------  -----------------------------------
    LOAD          003      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          004      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          005      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          006      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          009      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          010      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          011      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          012      +00000000    Success.
  ------------  -------- -----------  -----------------------------------
    LOAD          013      +00000000    Success.
```

```
------------  --------  -----------  ----------------------------------
  RESULTS:       9 of 9 LOADs completed successfully.
------------  --------  -----------  ----------------------------------

Summary of LOAD Agents:
Number of rows read          = 0
Number of rows skipped        = 0
Number of rows loaded        = 0
Number of rows rejected       = 0
Number of rows deleted        = 0
Number of rows committed      = 0
```

5.6　OLAP 系统设计与应用开发最佳实践

5.6.1　表的设计最佳实践

- 分区键决定了表的数据如何在节点间进行分步，选用在表关联中出现最多且唯一值很多的字段，如账号、卡号、客户号等。在数据分布均匀的前提下，分区键中包含的字段数越少越好，避免不必要的网络开销。如无特殊情况，建议只选用一个字段作为分区健。
- 相同业务含义的字段，尽量采用相同的数据类型、长度、精度，特别是分区键字段；否则在进行表关联的时候需要使用类型转换和长度截取函数，这些计算函数往往会造成全表扫描。
- 对小数据量(1 万以下，比如码表、维表)的表，尽量建立在单分区数据库节点中，其中经常参与表关联的，建议复制到每个分区，这样可以减少数据广播带来的网络成本。
- 另外原表上的索引尽量迁移到复制表上创建，这样可以提高 SQL 查询使用复制表的概率。
- 用户临时表需要指定分布键，否则 DB2 采用第 1 个字段作为分区键，这样往往会导致效率低下，还会经常遇到临时表空间不足的情况；用户临时表也需要执行 RUNSTATS，否则后续 SQL 执行计划可能不好，导致性能低下。
- 合理使用范围分区，范围分区数不要太多，总数控制在 200 个以内比较好；尽量使用 PARTITIONED INDEX，如果有大量简单查询(仅返回数条记录)且查询条件无范围分区字段条件，则考虑 NOT PARTITIONED INDEX。
- 合理使用 MDC 技术，MDC 键中的字段组合值不能太多，否则会导致空间的浪费，数据加载时也比较慢；如果使用了 MDC 后空间消耗变大 20%以上，建议使用

GENERATED COLUMN 将 MDC 字段粒度加粗，如原来按日期字段建的 MDC，可以增加取“月份”后建成 MDC。

- 表字段尽量定义为 NOT NULL WITH DEFAULT；事实表必要时才定义主键，维表需要定义主键，事实表和维表间建立外键约束关系，使用 NOT ENFORCED ENABLE QUEYR OPTIMIZATION 选项。
- 建议总是收集分布式统计信息(WITH DISTRIBUTION)。
- 如果采用自动收集统计信息，建议注册收集统计信息的方法，如 RUNSTATS ON TABLE TABLE_SCHEMA.TABLE_NAME WITH DISTRIBUTION AND SAMPLED DETAILED INDEXES ALL SET PROFILE ONLY。

5.6.2 数据访问方式最佳实践

- 大表操作时，尽量减少用户自定义函数的使用，尽量使用 DB2 提供的函数，用户自定义函数无法利用 DB2 的分区并行特性。如果自定义函数中还查询了别的表，最好将 SQL 中自定义函数中的表拿出来，改写成表关联的形式。
- WHERE 条件中，日期时间字段的判断，尽量不使用 TO_DATE 等函数，直接使用时间格式的字符串，如 ETL_LAD_DTE> '2013-02-19'，而不是 ETL_LAD_DTE> TO_DATE('2013-02-19', 'YYYY-MM-DD')。
- 大表关联的条件，最好能写成等值判断，如 WHERE LOCATE(A.COL1, B.COL2) = 1，改写成 WHERE A.COL1=SUBSTR(B.COL1, 1, LENGTH(A.COL1))。
- 两张大表 TAB_1 和 TAB_2 关联，关联条件为多个 OR 条件，且其中一个是 EXISTS/NOT EXISTS，建议将 EXISTS/NOT EXISTS 改写成 LEFT JOIN。
- 尽量不使用游标，尽量将游标改写成表关联，DPF 环境游标操作数据会经过协调节点，效率比较低。
- 尽量减少开发工具长期连接数据库，如 TOAD 可能频繁出现异常现象，从 TOAD 到数据库的连接数增加到 2000 个，直接造成数据库服务器 hung 住，ETL 作业无法运行；另外，TOAD 工具加异常系统表锁，直接导致大量的 ETL JOB 处于锁定等待状态，而不是运行状态，这样的状态会给开发人员或数据库使用人员造成“数据库慢”的假象。
- 对于一些大量的汇总数据操作，我们建议尽早进行汇总操作，否则最后的数据量会非常大。
- 一定要杜绝跨分区组的关联查询操作(单分区小表复制除外)，多分区跨分区组的关联查询操作一定会造成大量的数据广播，给分区间网络通信造成很大的负担。

- 如果 WHERE 条件中作为“字段等值判断”的字段唯一值较多，建议调大 num_freqvalues；如果“非等值”判断或 Like，建议调大 num_quantiles。可以使用调数据库参数的方式，也可针对这些表在 RUNSTATS 时调大这两个参数。RUNSTATS 需要使用 WITH DISTRIBUTION 选项。
- 如果 WHERE 条件中，一个表上有多个字段进行谓词判断，建议收集这些字段的分组统计信息(GROUP STATISTICS)。
- 大批量数据处理的存储过程最好设置为 REOPT ALWAYS，或执行前重新编译，保证比较好的执行计划。
- 如果主要是大批量计算，不涉及小数据集查询(如根据账号查询交易明细)，建议删除相关索引。
- WHERE T2.LAST=T2.FIRST+1，且 T2.FIRST 上有索引，建议修改为 WHERE T2.LAST－1 =T2.FIRST，以利用 T2.FIRST 上的索引。
- 表关联时，如果经常使用表达式进行关联，如 A.EVENT_ID='SSP'||replace (char(B.etl_lad_dte),'-','')||trim(B.pay_biz_seq_num)，建议将表达式定义成 B 表的 GENERATED COLUMN。

5.6.3　复制表的定义

经常参与关联的小表建议放到单分区，并且创建复制表到每个分区节点，以减少网络广播。创建复制表的语法如下：

1. 源表结构

```
CREATE TABLE "DB2UAPP "."TEST"  (
                  "REPT ID" VARCHAR(30 OCTETS) NOT NULL ,
                  "SHOW ID" VARCHAR(3 OCTETS) NOT NULL ,
                  。。。 。。。
                  "INVALID DTE" DATE )
                 IN "APP DATA DAT S" INDEX IN "APP DATA IDX S"
                  ORGANIZE BY ROW;
```

2. 创建复制表，建表语句如下

```
CREATE TABLE DB2UAPP.RP TEST
AS  (
  SELECT *
  FROM DB2UAPP.TEST
) DATA INITIALLY DEFERRED
  REFRESH  DEFERRED
```

```
ENABLE QUERY OPTIMIZATION
DISTRIBUTE BY REPLICATION
IN "APP MA MD DAT"
INDEX IN "APP_MA_MD_DAT_IDX";
```

注意：

复制表的表空间要指定多分区的表空间。

3. 刷新复制表

```
refresh table DB2UAPP.RP_TEST;
```

注意：

1) DFT_REFRESH_AGE 参数需要配置成 99999999999999，需要重启数据库才生效，否则 SQL 里必须显式执行 SET CURRENT REFRESH AGE ANY，才可以使用该 MQT。

2) 每次更新完主表，都需要刷新复制表才能生效，否则仍然会使用老数据，建议在有参数表更改的时候进行刷新，或者在当天跑批完成后统一刷新。

3) refresh immediate 的复制表需要源表有唯一索引才能创建。

5.7 DB2 列组织表

5.7.1 DB2 列组织表介绍

DB2 V10.5 引入了列组织表，为 OLAP 系统的设计提供了新的选项。

从 DB2 V10.5 开始，可以创建采用按列组织进行存储的列组织表。不仅是数据组织方式的变化，同时还有对内存、CPU 和 I/O 等多方面的优化，能够更好支持具有复杂查询的分析型工作负载，这些新功能统称为 BLU 加速。BLU 加速易于进行设置和自行优化，不需要索引和聚集或者耗时的数据库调整就能实现较高的性能和存储效率。

需要注意的是，对于 OLTP 工作负载，不应该使用列组织表，使用索引访问的传统行组织表通常更适用于 OLTP 应用。

DB2 V10.5 中新提供了以下相关内容：

- DB2_WORKLOAD 可以设置为 ANALYTICS，用于启用列组织、自动执行的专用初始内存配置、页面和扩展数据块大小配置、空间回收以及自动工作负载管理。
- CREATE TABLE 语法中增加了新的子句“ORGANIZE BY COLUMN”来创建列组织表。
- 增加了新的数据库配置参数(dft_table_org)，用于更改默认的表组织方式。
- 增加了新的 db2convert 实用程序，用于将行组织表转换为列组织表。

- REORG TABLE 命令可以支持回收列组织表使用的存储空间。
- 自动执行的工作负载管理，可以显著提高同时运行多个查询的工作负载的性能和系统使用率。
- 动态列表预取，这是一种在访问列组织表的查询执行计划中使用的新预取类型。
- 支持 NOT ENFORCED 主键和唯一约束，可以在知道数据已经符合约束时避免性能开销和空间需求，并且支持列组织表上的 ENFORCED 主键和唯一约束。

5.7.2　DB2 列组织表应用场景和环境配置

如果工作负载完全是分析型或 OLAP 工作负载，那么建议尽可能使更多的表采用列组织格式。如果数据库中的大部分表都将使用列组织表，那么可以在创建数据库之前将 DB2_WORKLOAD 注册表变量设置为 ANALYTICS。这样做有助于配置内存、表组织、页大小和扩展数据块大小，并且会启用工作负载管理。

DB2 优化了向列组织表大量插入数据或者大量更新已有数据的事务处理。因此，对于事务型工作负载，不应该使用列组织表，反而使用索引访问的传统行组织表通常更适用于这些类型的工作负载。针对列组织表，在每个插入或更新事务影响 100 行或更多行的情况下，性能最佳。

对于包含分析型查询处理与具有很高选择性的访问(涉及不到 2% 的数据)的组合的混合工作负载，行组织表与列组织表的混合可能更适用。

1. “一键式”地创建和配置数据库

创建完全是分析型或 OLAP 工作负载的数据库，使用的命令如下：

```
db2set DB2 WORKLOAD=ANALYTICS
db2start
db2 create db mydb
```

这种“一键式”的创建和配置方法非常简单，只需要配置 DB2_WORKLOAD 注册表变量为 ANALYTICS，来告诉 DB2 这是一个为 OLAP 系统创建的数据库，其他的事情 DB2 会自动处理。表 5-1 列出了 DB2 在创建数据库期间自动配置的部分内容。

表 5-1　DB2 创建分析型数据库时的配置

参数类型	参数设置
一般数据库参数	DFT_TABLE_ORG = COLUMN PAGESIZE = 32KB DFT_EXTENT_SZ = 4 DFT_DEGREE = ANY

(续表)

参数类型	参数设置
数据库内存配置	CATALOGCACHE_SZ、SORTHEAP 和 SHEAPTHRES_SHR 被调整为比默认值更大的值，并根据硬件配置选择最优的配置
内部并发参数配置	即使数据库实例参数 INTRA_PARALLEL 为 OFF 的情况，也为所有 MAXIMUM DEGREE 设置为 DEFAULT 的工作负载启用内部并发机制
DB2 工作负载管理(WLM)	SYSDEFAULTMANAGEDSUBCLASS 服务子类中的 threshold SYSDEFAULTCONCURRENT 被激活
自动空间回收	设置 AUTO_MAINT = ON 和 AUTO_REORG = ON，并且默认对列组织表进行空闲空间的回收

注：表 5-1 中的 SYSDEFAULTMANAGEDSUBCLASS 是 BLU 加速功能中新加入的服务子类。

对于已有的数据库，也可以通过以下命令对其重新配置：

```
db2set DB2 WORKLOAD=ANALYTICS
db2start
db2 connect to mydb
db2 autoconfigure apply db
```

DB2_WORKLOAD 是实例级别的全局注册变量，设置该变量会影响所有新创建的数据库。如果一个实例下有多个数据库，其中部分数据库不用于 OLAP 型工作负载，可以只在创建特定数据库之前设置该注册变量，创建完成之后再设置该注册变量为空。

2. 手工创建和配置数据库

如果在某些情况不允许设置 DB2_WORKLOAD 注册表变量为 ANALYTICS 以自动对数据库进行配置，也可以手工创建数据库，并以最优方式针对分析型工作负载进行配置。

要手动为分析型工作负载创建和配置数据库，需要执行以下操作：

(1) 创建具有 32KB 页大小、UNICODE 代码集(默认值)以及 IDENTITY 或 IDENTITY_16BIT 整理的数据库。命令如下：

```
CREATE DATABASE DMART USING CODESET UTF-8 TERRITORY US COLLATE USING IDENTITY
PAGESIZE 32 K
```

(2) 确保 sheapthres 数据库管理器配置参数设置为 0(默认值)。请注意，此设置适用于实例中的所有数据库。

(3) 更新数据库配置，如下所示：

- 将 dft_table_org(用户表的默认表组织)数据库配置参数设置为 COLUMN，以便默认情况下新表创建为列组织表；否则，必须在每个 CREATE TABLE 语句上指定 ORGANIZE BY COLUMN 子句。
- 将 dft_degree(默认等级)数据库配置参数设置为 ANY。
- 将 dft_extent_sz(默认扩展数据块大小)数据库配置参数设置为 4。
- 将 catalogcache_sz(默认高速缓存)数据库配置参数的值增大 20%(它是在数据库创建期间自动设置的)。
- 确保 sortheap(排序堆)和 sheapthres_shr(共享排序的排序堆阈值)数据库配置参数没有设置为 AUTOMATIC。考虑为分析型工作负载显著增大这些值。合理的起始点是将 sheapthres_shr 设置为缓冲池的大小(在所有缓冲池上)。将 sortheap 设置为 sheapthres_shr 的某个尾数(例如，1/20)以启用并行排序操作。
- 将 util_heap_sz(实用程序堆大小)数据库配置参数设置为 1 000 000 页和 AUTOMATIC 以满足 LOAD 命令的资源需求。如果数据库服务器至少具有 128GB 内存，可以将 util_heap_sz 设置为 4 000 000 页。如果并行装入操作正在运行，也可以临时增大 util_heap_sz 的值以满足更高内存需求。
- 将 auto_reorg(自动重新组织)数据库配置参数设置为 ON。

(4) 确保启用查询内并行性，这是访问列组织表所需要的。可以在实例级别、数据库级别、工作负载级别或应用程序级别启用查询内并行性。

(5) 启用 SYSDEFAULTMANAGEDSUBCLASS 服务子类上的并行控制，命令如下：

```
ALTER THRESHOLD SYSDEFAULTCONCURRENT ENABLE
```

5.7.3　创建列组织表

创建列组织表的命令非常简单，下面的示例说明了如何创建一个列组织表：

```
CREATE TABLE mytable (
c1 INTEGER NOT NULL,
c2 INTEGER,
…
)
ORGANIZE BY COLUMN IN <table_space_name>
```

可以看到，创建列组织表和普通的行组织表的唯一不同在于 ORGANIZE BY COLUMN 子句。如果数据库参数 DFT_TABLE_ORG 设置为 COLUMN，就不需要指定 ORGANIZE BY COLUMN 子句。

除此之外不要任何其他指定。在上面的示例中，创建列组织表时指定了表空间，建议将事实表放在单独的表空间中，将维度表放在另外的表空间中。如果还要创建 ENFORCED 类型的主键约束或唯一约束，建议使用 INDEX IN 为索引指定单独的表空间。

另外，建议总是指定 NOT NULL，除非明确需要存放 NULL 值。因为在列组织表中可为空的列实际上在内部是两个列，其中一列存放可为空的属性值，另一列存放真正的值。

对于已经存在的表，可以使用以下语句来查看该表是列组织表还是行组织表：

```
SELECT tabname, tableorg, compression
FROM syscat.tables
WHERE tabname like 'SALES%';
TABNAME                         TABLEORG COMPRESSION
------------------------------- -------- -----------
SALES COL                              C
SALES ROW                              R           N
2 record(s) selected.
```

如果在 syscat.tables 中，该表的记录中 TABLEORG 为'C'，则是列组织表；如果 TABLEORG 为'R'，则是行组织表。

对于列组织表，也可以像行组织表一样创建主键约束或唯一约束。在 DB2 V10.5 中增加了 ENFORCED 和 NOT ENFORCED 选项，对于 ENFORCE 类型的主键约束或唯一约束，DB2 会创建相应的索引；而对于 NOT ENFORCED 类型的主键约束或唯一约束，DB2 并不会创建相应的索引，其用途是告诉 DB2 优化器这个约束信息以便优化器作出最优的执行计划。这些特性同样适用于列组织表。

列组织表在创建时，DB2 还会在内部默认为每个列组织表创建一个概要表(synopsis table)，这个概要表里保存了列组织表所有的非字符类型的字段，例如时间戳、布尔型或数字类型的字段。如果有主键约束或唯一约束，约束索引的字段值也放在概要表中。概要表中还分段记录了每一列的最大值、最小值信息，用于在查询时快速过滤数据。

可以通过查询 syscat.tables 查询列组织表对应的概要表，查询语句如下：

```
SELECT tabschema, tabname, tableorg
FROM syscat.tables
WHERE tableorg = 'C';
TABSCHEMA    TABNAME                         TABLEORG
------------ ------------------------------ --------
DB2INST1     SALES COL                        C
SYSIBM      SYN130330165216275152 SALES COL C
2 record(s) selected.
```

5.7.4　向列组织表装入(LOAD)数据

由于列组织表通常用于 OLAP 系统，因此列组织表中的数据通常是从行组织表中装入或批量插入的。向列组织表装入数据的过程，数据会被转换为按列组织并且自动进行数据压缩，因此列组织表通常比行组织表要小很多，这不仅得益于高效的压缩算法，同时也因为减少了辅助信息(例如索引等)。

在对列组织表进行初始装入数据时，DB2 会自动创建列压缩字典。在后续进行数据装入或批量插入时，DB2 还会创建数据页级别的压缩字典，对数据进行进一步的压缩。初始装入的数据直接影响列压缩字典的质量，因此建议初始装入的数据是所有数据的典型抽样。

向列组织表装入数据的命令与行组织表所用的命令完全相同，但是在 LOAD 的内部过程上还是有一些不同之处：

- 向列组织表装入数据新增加了 ANALYZE 阶段
- 数据被转换为按列组织并进行了压缩
- 维护概要表(synopsis table)

表 5-2 介绍了向列组织表装入数据的内部阶段。

表 5-2　向列组织表装入数据的内部阶段

阶段	描述
ANALYZE	只有在创建压缩字典时才需要该阶段。这个阶段内 DB2 建立柱状图来记录所有列中数据的出现频率，并相应建立压缩字典
LOAD	用列压缩字典和数据页压缩字典来压缩数据，然后把压缩后的数据存入数据页，同时更新维护概要表
BUILD	如果有 ENFORCED 的主键约束或唯一约束，在这个阶段创建相应的唯一索引

为了更高的压缩比和更高效率的数据装入，需要考虑两个因素：

1) UTIL_HEAP_SZ 数据库参数直接影响在内存中能维护多少不同的数据值来创建压缩字典，从而影响压缩字典的质量。可以在初始装入之前，将 UTIL_HEAP_SZ 数据库参数设置的尽可能大，然后在装入完成之后把 UTIL_HEAP_SZ 改为正常的值。UTIL_HEAP_SZ 可以在线修改。

2) 如果可能的话，尽量对待装入的数据进行预先排序，例如可以按照最常用的过滤事实表的查询谓词所用的字段进行排序，或者按照事实表与维度表进行关联查询时的关联字段。对数据预先排序之后再装入列组织表，可以提高压缩比并且提高数据查询的性能。

要减少数据装入的时间，可以只用一部分数据来创建压缩字典，从而减少 ANALYZE 阶段的时间。这个方法的步骤如下：

(1) 创建列组织表，但不要创建任何主键约束或唯一约束。

(2) 从待装入的数据中获取一部分有代表性的数据，如果待装入的数据存放在别的 DB2 数据库中，可以用抽样查询的游标获取抽样数据。

(3) 用抽样数据建立压缩字典，命令如下：

```
LOAD FROM subset data.del OF DEL
REPLACE RESETDICTIONARYONLY INTO mytable;
```

(4) 装入所有数据，命令如下，这个过程不需要 ANALYZE 过程：

```
LOAD FROM all_data.csv OF DEL INSERT INTO mytable;
```

(5) 创建 ENFORCED 或 NOT ENFORCED 的主键约束或唯一约束。

(6) 执行 runstats。

需要注意的是，这个方法可以减少数据装入的时间，但如果抽样数据不够典型，会影响压缩效果。

5.7.5 列组织表的访问计划

DB2 BLU 加速简化了对列组织表的访问方式，通常只考虑表扫描、hash join 和 hash 聚集。表连接的次序依然是根据成本(cost)来选择，基数(cardinality)估算信息尤其重要，因此和行组织表类似，也需要定期收集统计信息。

创建 NOT ENFORCED 主键约束或唯一约束对于提高列组织表的查询性能非常有帮助，这能够给 DB2 优化器提供更多的信息，以便优化器选择最优的执行计划。

列组织表的访问计划中增加了新的 CTQ 操作符，用来指示列组织表和行组织表之间的转换过程。在执行计划中，所有 CTQ 操作符下方的操作符都针对列组织表进行了优化，因此对于列组织表来说，一个好的执行计划应该是绝大多数操作符都在 CTQ 操作符的下方，例如图 5-13 中的执行计划。

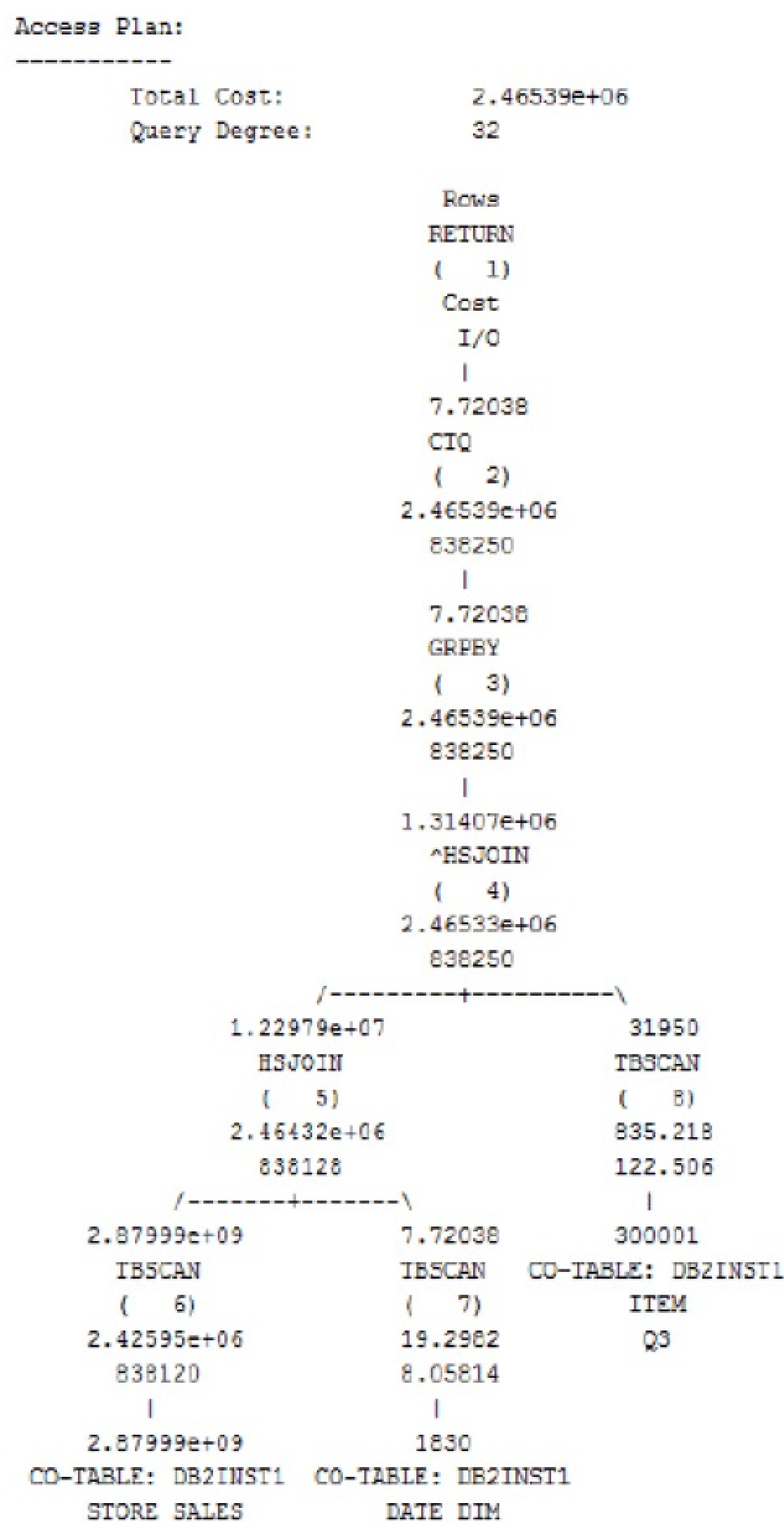


图 5-13　列组织表的访问计划

5.8　本章小结

本章我们主要讲解了 OLAP 系统的设计和管理，介绍了 OLAP 系统常用的 DB2 组件，如 DB2 DPF 和最新的 BLU 加速技术。对于 DB2 DPF，介绍了其基本架构和概念，还介绍了一种常用的利用 DPF 的 OLAP 高性能设计，较为详细地讲解了 DPF 配置和常见运维操作实践案例，最后提供了基于 DPF 架构的 OLAP 系统设计与应用开发最佳实践。DB2 列组织表以及整个 BLU 加速技术，为设计 OLAP 系统提供了新的选择。本章介绍了列组织表的相关概念、配置方法、如何使用列组织表以及列组织表的访问计划。

第 6 章

高可用与灾备

作为企业的关键业务软件系统，往往需要能够在发生各种异常事件时快速恢复可用性，以保证业务的连续性，这样才能最低程度地降低异常事件带来的损失。

我们想象一下，如果美国 9•11 事件发生的时候，世贸大楼中的数据库系统没有异地的灾备方案，那么所有的企业数据将会丢失，这对企业来说无疑是一次灾难。多数情况下，这种损失是无法挽回的。统计表明，在丢失企业关键数据后，有 50%以上的企业会因此而关门歇业。

为了应对上面的挑战，各种高可用技术应运而生，而 DB2 HADR 技术正是在数据库层应对异常事件、最大程度保证数据库可用性的技术。

本章主要讲解如下内容：

- HADR 的设计理念
- HADR 典型场景的搭建
- HADR 的维护
- HADR 性能调优
- HADR 高可用案例分享

6.1 HADR 的设计理念

6.1.1 什么是高可用性

高可用性(High Availability)是各行业对 IT 产品的一项常见需求。随着 IT 系统对于企业的重要性越来越高，意外的系统宕机对企业造成的损失越来越大，企业对此类问题的容忍空间也越来越小，这就使各种高可用性技术应运而生。例如 IT 硬件设施的高可用性设计(如冗余的电源等、冗余的各种硬件模块等)、OS 层次的高可用性(如 AIX 的 PowerHA、HPUX 的 ServiceGuard、Linux 的 HeartBeat、Windows 的 MSCS 等)、中间件的高可用性(如各种集群技术)、网络的高可用性、数据库的高可用性(DB2 的 HADR、Informix 的 HDR 和 Oracle 的 Data Guard 等)等。通常，企业会根据自身的需求和投入情况，综合考虑来应用不同的高可用性技术，进而提供最符合自己需求的高可用性方案。

在这里有一点值得注意的是，高可用性方案专指提供高度可用性的方案，而不是完全可用性，也就是 100%可用的方案。因此一般称为高可用性的技术方案，通常意味着在极端场景下存在着较小概率的不可用发生。

什么是 HADR

HADR 是 High Availability Disaster Recovery 的缩写，从字面的意思可以解释为高可用灾难恢复；从名称可以看出，这项技术的目的是提供灾难恢复场景中用到的高可用性能力。

HADR 就是 IBM 的 DB2 产品在灾备场景下的高可用解决方案。当然随着需求场景的不断丰富，HADR 技术已经不局限于灾备场景的应用，而是可以应用于很多场景的基于 DB2 数据库的高可用方案。

HADR 在 DB2 V10.1 版本时有较大增强，支持 MULTI-STANDBY(一主多备)，后面会有讲解。

高可用的对象

高可用功能提供的高可用能力当然是指在某些故障发生时，服务不会因为故障而停止，高可用功能会保障服务在不中断或短暂中断后继续能够正常使用。而造成服务可能中断的场景，我们将之分为两类：计划性中断和非计划性中断。

- 计划性中断：计划性中断指的是服务的停止是基于事先的计划、按照既定的步骤实施的结果，通常的系统维护都属于这个范畴，如升级、打补丁、重启等。计划性中断通常都处于可控的范围，在需要计划性中断时，采用高可用功能可以快速完成服务的切换，缩短服务的中断时间，大大提高计划性维护工作的效率。

- 非计划性中断：非计划性中断指的是由于意外故障导致服务的中断，在这种情况下，高可用功能一般可以通过手工方式或自动检测到发生的故障，并采用事先设置好的方式，将服务及相关资源转移到冗余环境下以继续提供服务。此时，高可用功能会非常好地解决服务中断的问题，并会高效地继续提供服务。

DB2 的 HADR 技术，正是在 DB2 数据库级别提供了上述能力。但是值得注意的是，当故障发生时，HADR 技术本身并不能自动发生切换，而需要手工执行这一操作。在很多时候，这样做会导致效率下降，而为了弥补这一不足，通常需要与其他的 OS 层次的高可用技术结合，如 IBM 的 TSA 和 PowerHA 产品。在本章的最后部分，我们会讨论相关内容。

6.1.2　HADR 的原理

为了完成 DB2 数据库的高可用能力，HADR 中定义了不同的角色及相关的行为。下面我们分别介绍 HADR 最基本的行为规则。

HADR 最基本的行为规则

两台服务器构成两个节点(DB2 V10.1 版本开始支持多 STANDY)，一个节点作为 PRIMARY，另一节点作为 STANDBY。

- PRIMARY 节点：用于提供数据库的正常访问，处理在数据库中提交的事务。同时将数据库中产生的新日志通过网络发送至 STANDBY 节点。
- STANDBY 节点：通过恢复 PRIMARY 节点的数据库备份而创建；接收并保存 PRIMARY 节点发送过来的日志；通过 redo PRIMARY 节点发送过来的日志，复制 PRIMARY 节点的所有数据；DB2 V9.7 版本及以后，STANDBY 可以提供只读访问。

当 PRIMARY 节点发生故障时，STANDBY 节点会转换为 PRIMARY 节点，并继续提供数据库的访问。由于原 PRIMARY 端的数据已经通过 redo 日志的方式传递到了原 STANDBY 端，因此当原 STANDBY 端变为新的 PRIMARY 端时，数据库中的数据保持与原 PRIMARY 端的一样。对于数据库的客户端来说，两者提供的数据库服务是没有区别的。

当故障的 PRIMARY 节点从故障中恢复后，会转换为 STANDBY 节点，并与新的 PRIMARY 进行数据同步，从而成为新的 HADR 节点对。

原理总结如图 6-1 所示。

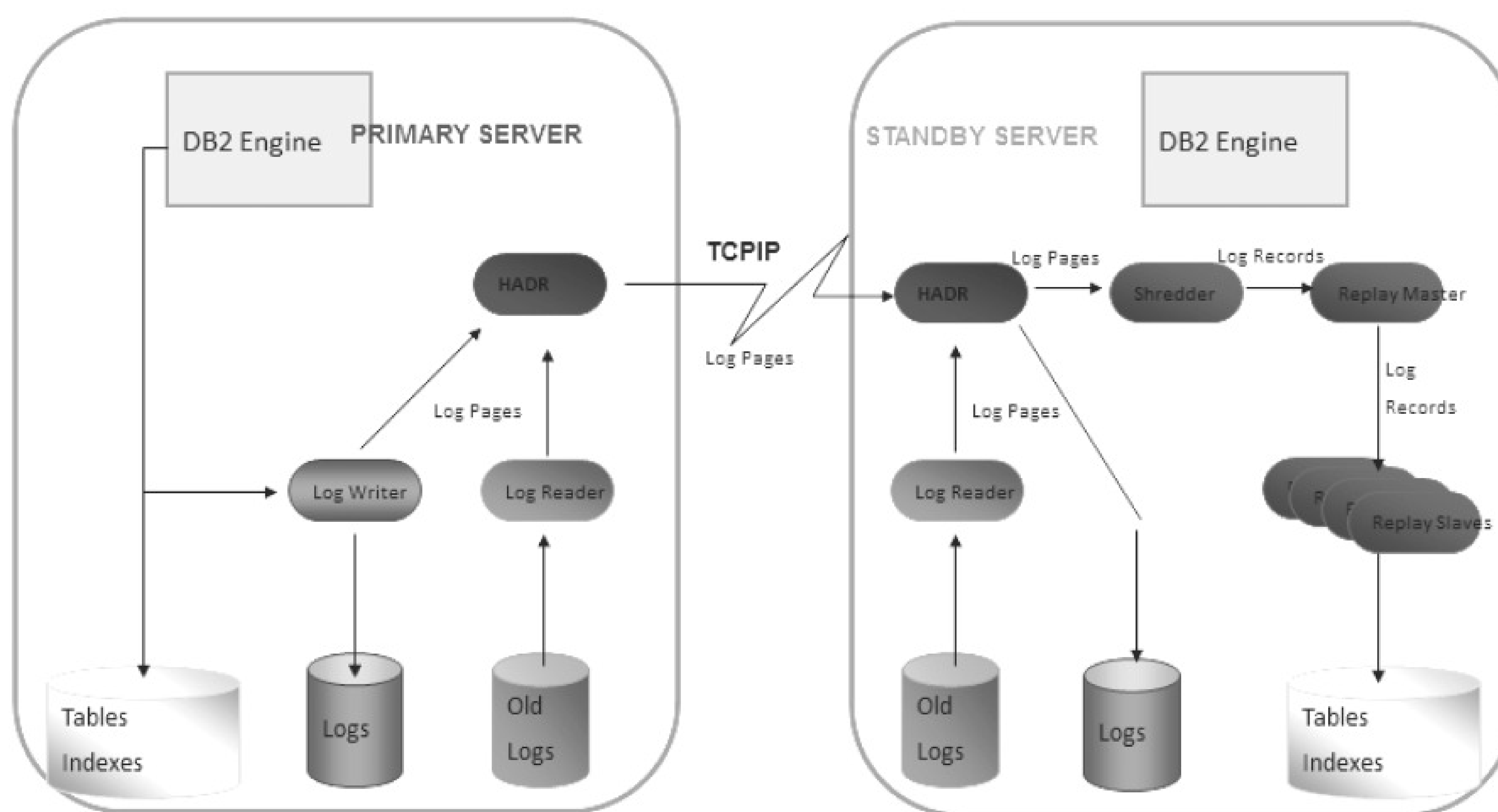


图 6-1　DB2 HADR 原理

6.1.3　HADR 的日志处理模式

从上面的图 6-1 可以看出，在 PRIMARY 端的所有日志都需要通过网络发送到 STANDBY 端。为了保持 STANDBY 端数据与 PRIMARY 端数据的同步，HADR 提供了几种不同的日志处理模式，在不同的模式下体现了不同的同步级别，而不同的同步模式会影响整个 HADR 提供的高可用能力。其中，SYNC 模式提供了最好的高可用能力，最大程度地保证了数据一致性，但性能会受到影响，其他模式会在逐渐降低高可用能力的条件下提高性能。

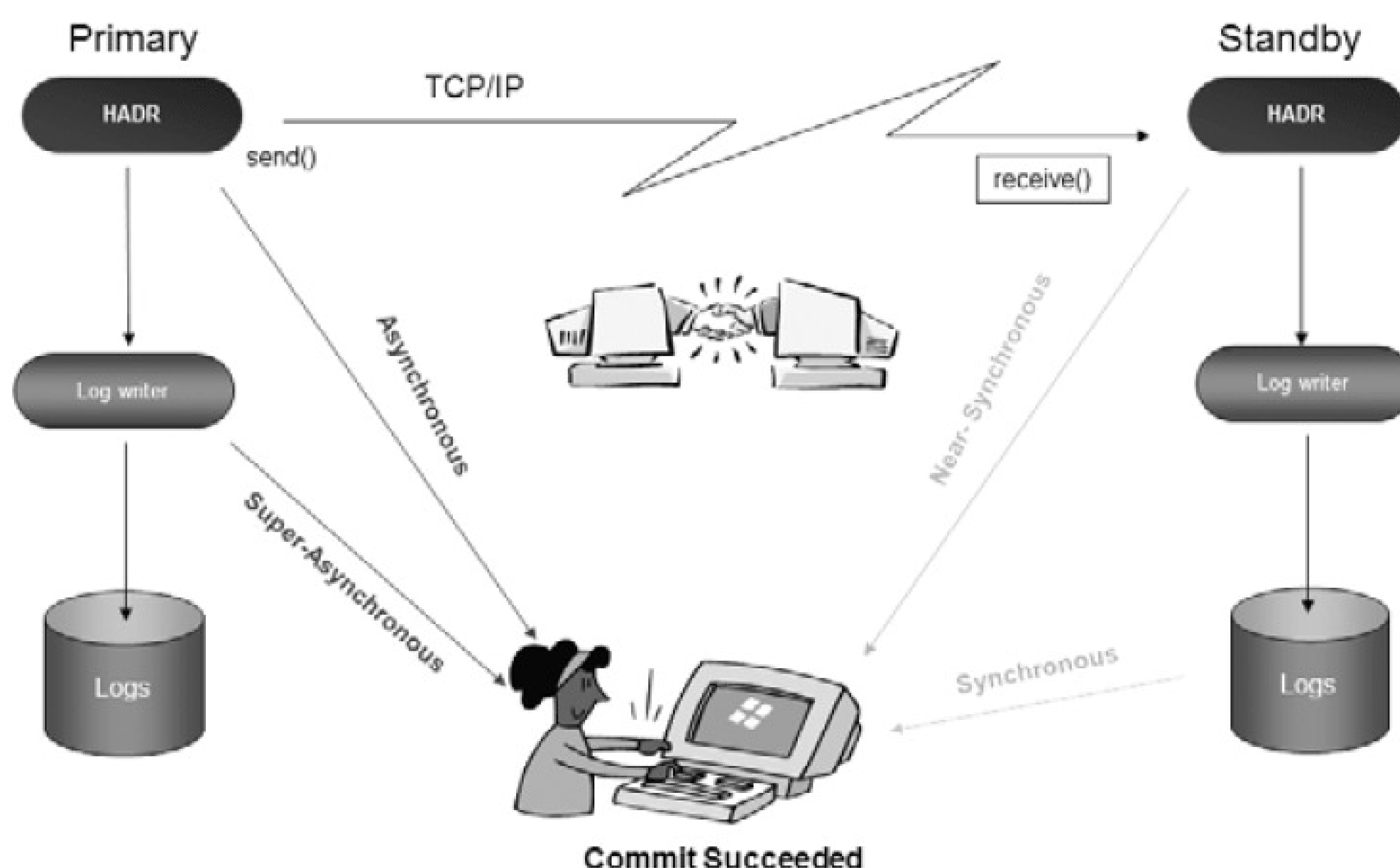


图 6-2　DB2 HADR 日志的处理模式

DB2 HADR 提供的同步模式如下：

- SYNC 模式

SYNC(同步)模式的日志处理要求如图 6-2 所示的 Synchronous 箭头所指，这种模式要求在 STANDBY 端接收到日志并且由 Log writer 线程将日志写入磁盘后，才会给 PRIMARY 发送应答，而 PRIMARY 只有在接收到这一应答后，才会允许日志相关的事务完成 COMMIT，在收到应答之前相关的 COMMIT 是被阻塞的。因此，SYNC 模式保证任何日志都不会在数据同步过程中丢失，从而保障 PRIMARY 和 STANDBY 两端数据的最大一致性。

但由于日志从 PRIMARY 传输到 STANDBY，接收后再写入磁盘，此过程需要的时间较长，这就导致 PRIMARY 端的每个 COMMIT 需要等待的时间相对较长，最终影响事务的性能。因此，SYNC 模式以牺牲性能为代价，在最大程度上保障了数据的一致性。

- NEARSYNC 模式

NEARSYNC(接近同步)模式的日志处理要求如图 6-2 所示的 Near-Synchronous 箭头所指，这种模式要求在 STANDBY 端的 HADR 处理线程在内存中收到日志后，即向 PRIMARY 发送应答，在 PRIMARY 端接收到应答后，才允许日志相关的事务完成 COMMIT。因此，NEARSYNC 保证日志一定是已经发送到了 STANDBY 端，但无法保证异常情况下 STANDBY 端的日志丢失，这已经在很大程度上对日志发送的可靠性提供了保障。

由于在 NEARSYNC 模式下，日志在写入磁盘之前即向 PRIMARY 发送了应答，因此在很大程度上提高了日志发送/应答的效率，也就是 COMMIT 完成前等待 STANDBY 应答的时间缩短了，从而使 PRIMARY 端的数据库性能得到了提升。

- ASYNC 模式

ASYNC(异步)模式的日志处理要求如图 6-2 所示的 Asynchronous 箭头所指，这种模式要求 PRIMARY 端的 HADR 线程在发送日志后，即刻允许日志相关的事务 COMMIT，而不管 STANDBY 端是否能够成功接收到日志。因为这种模式下 PRIMARY 端无须等待任何来自 STANDBY 端的应答，所以此模式下 PRIMARY 端的数据库性能会很好。

因为 ASYNC 模式不需要等待 STANDBY 端的应答，在提升数据库性能的同时，却降低了 HADR 架构的可靠性，所以在发生故障时 STANDBY 端丢失事务的风险显著增加了。

- SUPERASYNC 模式

SUPERASYNC(超级异步)模式是 DB2 V10.1 版本新增加的功能，这种模式对日志处理的要求如图 6-2 所示的 Super-Asynchronous 箭头所指，这种模式要求 PRIMARY 端的本地 log writer 线程写日志成功后即允许日志相关的事务 COMMIT，这种模式的日志处理要求与非 HADR 架构下的单机 DB2 数据库的处理方式类似，事务的 COMMIT 不会因为使用了 HADR 技术而有任何延时，因此性能最好，但同时这种模式下在 PRIMARY 端有故障时丢

失事务的风险也最高。

综合考虑上述各种模式的特点，我们总结一下各种模式的适用场景。

- SYNC 模式适用于对性能要求不高，但对数据库中数据的一致性要求极高的场景，在任何情况下 PRIMARY 端发生故障后，都不允许出现事务的丢失，即使因为日志的传输而导致 PRIMARY 端的事务被阻塞也在所不惜。
- NEARSYNC 模式适用于对性能有一定要求，且对数据库中数据的一致性要求也较高的场景。为了保证数据库中数据的一致，可以适当阻塞 PRIMARY 端的事务。但是一般在设置的时限内，如果还无法完成事务日志的同步，就不再阻塞 PRIMARY 端的事务 COMMIT。
- ASYNC 模式适用于对性能要求较高，且对数据库中数据的一致性要求不高的场景，PRIMARY 端的事务会因为 HADR 线程发送日志有一点延时，但延时很短。
- SUPERASYNC 模式适用于对性能要求极高，且对数据库中数据的一致性要求不高的场景，PRIMARY 端的事务不会因为 HADR 功能而有任何延时。

6.1.4 HADR 的限制

并非DB2数据库中发生的所有行为都能够通过HADR技术从PRIMARY复制到STANDBY端，由于 HADR 技术的本质是通过 redo 日志的方式来实现数据库的同步，因此受此限制很多行为是无法复制到 STANDBY 端的。

HADR 不能支持的操作有：

- 不记录日志的所有操作和对象。
- 使用了无穷日志设置。
- 在 STANDBY 端备份日志。
- 恢复的数据库或表空间。
- 使用了 COPY NO 选项的 LOAD 操作。

其他限制还有：

- HADR 是 ESE 版本自带的功能，其他版本需要单独的 license 支持。
- HADR 不支持 DPF 环境。
- HADR 要求 PRIMARY 端和 STANDBY 端有相同的操作系统版本和 DB2 版本。
- HADR 虽然在 DB2 V9.7 开始支持 STANDBY 的只读操作，但不支持在 STANDBY 端执行数据库的 BACKUP。
- HADR 无法自动诊断 PRIMARY 发生的故障，这需要由其他的高可用软件完成。

6.2　HADR 典型场景的搭建

在实施 HADR 的方案时，由于不同企业在不同环境下可能有很多不同的特点，最终导致实现的 HADR 可能会千差万别，在本书很难穷尽这些场景。为了突出 HADR 最基本的设计理念，我们在本节只讨论一下典型 HADR 环境下的搭建要求和搭建过程。

6.2.1　对基础环境的要求

HADR 的设计初衷是为了保证在 PRIMARY 端发生任何故障时，STANDBY 都可以转换为 PRIMARY 端以继续提供数据库服务。为了实现这一点，基础环境方面必须满足一定的要求。

- 操作系统方面，需要保证 PRIMARY 端和 STANDBY 端的操作系统版本一致。
- 服务器方面，虽然原理上 HADR 支持在同一台物理服务器上搭建 HADR，但是生产中为了彻底隔离服务器故障的影响，PRIMARY 与 STANDBY 必须部署在不同的物理服务器上。
- 存储方面，一般为了隔离存储故障，PRIMATY 与 STANDBY 需要分布在不同的存储上，多数情况是位于异地的不同存储。
- 网络方面，一般为了隔离对外服务网络和 HADR 日志传输网络的故障，同时为了避免互相造成的性能影响，HADR 需要使用专用网络，甚至使用广域网络，而通常数据库提供服务器的网络为企业内部网络。

6.2.2　HADR 的配置参数

HADR 有关的数据库设置为：

```
HADR database role                                      = STANDARD
HADR local host name                  (HADR LOCAL HOST) = HADRDBA
HADR local service name                (HADR LOCAL SVC) = DBAPORT
HADR remote host name                (HADR REMOTE HOST) = HADRDBB
HADR remote service name              (HADR REMOTE SVC) = DBBPORT
HADR instance name of remote server  (HADR REMOTE INST) = db2hadr
HADR timeout value                       (HADR TIMEOUT) = 60
HADR target list                     (HADR TARGET LIST) =
HADR log write synchronization mode     (HADR SYNCMODE) = NEARSYNC
HADR spool log data limit (4KB)      (HADR SPOOL LIMIT) = 52428800
HADR log replay delay (seconds)     (HADR REPLAY DELAY) = 0
HADR peer window duration (seconds)  (HADR_PEER_WINDOW) = 0
```

HADR database role 不是可以修改的参数，只能用于显示当前数据库在 HADR 下面的角色，取值为 PRIMARY、STANDBY 或 STANDARD。其中，PRIMARY 表示当前的数据库角色是 HADR 架构中的 PRIMARY 端，STANDBY 表示当前的数据库角色是 HADR 架构中的 STANDBY 端，STANDARD 表示当前的数据库没有启用 HADR 功能，只是一个标准的数据库。

配置参数 HADR_LOCAL_HOST 用于指定在 HADR 环境下本机的主机名或 IP 地址，此 PRIMARY 和 STANDBY 端的设置肯定是不一样的，而且通常此地址是 HADR 的专用地址，只用于 HADR 两端的通信，不再负责其他类型的通信。

配置参数 HADR_LOCAL_SVC 用于指定 HADR 服务在本地使用的端口，取值可以是端口名称或端口号，此设置在 PRIMARY 和 STANDBY 两端必须唯一，一般建议此设置与另外一端的端口号成对使用，并且此设置就是另外一端的参数 HADR_REMOTE_SVC 的值。

配置参数 HADR_REMOTE_HOST 用于指定对方的主机名或 IP 地址，与配置参数 HADR_LOCAL_HOST 相对应。

配置参数 HADR_REMOTE_SVC 用于指定对方的提供 HADR 服务的端口号，与配置参数 HADR_LOCAL_SVC 相对应。

配置参数 HADR_REMOTE_INST 用于指定对方远程的 DB2 实例名称，在 HADR 配置下，PRIMARY 与 STANDBY 端的实例名称可以相同，也可以不同。

配置参数 HADR_TIMEOUT 用于指定 HADR 等待通信应答的超时时间，单位是秒。如果 HADR 两端出现通信异常，会导致 HADR 进入等待状态，在等待时间达到此参数指定的时间后，如果通信还未能恢复，那么 HADR 的状态会发生改变，进而表示 HADR 的两端进入了不对等(non-Peer)状态。

配置参数 HADR_TARGET_LIST 用于启用多 STANDBY 功能，可以指定最多三个主机端，格式为:host:port。如果不需要启动多 STANDBY 功能，那么无须设置此参数。

配置参数 HADR_SYNCMODE 用来设置日志复制模式。

配置参数 HADR_SPOOL_LIMIT 用于指定 STANDBY 端可以缓冲至磁盘的日志量，PRIMARY 端发送的日志会被缓冲至磁盘，而且 PRIMARY 端也不会因此被阻塞。如果 STANDBY 发生故障，那么恢复后会继续从磁盘读取这些日志进行 replay。此配置参数提供了更好的容灾能力。

配置参数 HADR_REPLAY_DELAY 用于启动延迟更新功能，可以指定 STANDBY 端的数据更新比 PRIMARY 端延迟多长时间。

配置参数 HADR_PEER_WINDOW 用于指定 HADR 的两端可以在异常情况下继续处于对等(peer)状态的时间。

6.2.3　复制 PRIMARY 数据库

HADR 环境下的 STANDBY 端数据库源于 PRIMARY 数据库，创建 STANDBY 端数据库的方法可以是数据库恢复、存储的镜像功能或是其他类似的方法。

需要注意的是，数据库在 STANDBY 恢复完成后，必须处于 rollforward pending 状态，否则无法在 STANDBY 端启动 HADR。

恢复完成后，还需要针对 STANDBY 的情况修改数据库中 HADR 的相关参数，使其符合 STANDBY 端的实际情况，如 HADR_LOCAL_HOST 和 HADR_LOCAL_SVC 等。

6.2.4　启动 STANDBY

在配置好 HADR 后，就可以按照 HADR 的要求启动 HADR。设置示例如下：

PRIMARY 端的设置：

```
HADR database role                                           = PRIMARY
 HADR local host name                        (HADR LOCAL HOST) = HADRDBA
 HADR local service name                      (HADR LOCAL SVC) = DBAPORT
 HADR remote host name                      (HADR REMOTE HOST) = HADRDBB
 HADR remote service name                    (HADR REMOTE SVC) = DBBPORT
 HADR instance name of remote server   (HADR REMOTE INST) = db2hadr
 HADR timeout value                          (HADR TIMEOUT) = 60
 HADR log write synchronization mode      (HADR SYNCMODE) = NEARSYNC
 HADR peer window duration (seconds)  (HADR_PEER_WINDOW) = 0
```

STANDBY 端的设置：

```
HADR database role                                           = STANDBY
 HADR local host name                        (HADR LOCAL HOST) = HADRDBB
 HADR local service name                      (HADR LOCAL SVC) = DBBPORT
 HADR remote host name                      (HADR REMOTE HOST) = HADRDBA
 HADR remote service name                    (HADR REMOTE SVC) = DBAPORT
 HADR instance name of remote server   (HADR REMOTE INST) = db2hadr
 HADR timeout value                          (HADR TIMEOUT) = 60
 HADR log write synchronization mode      (HADR SYNCMODE) = NEARSYNC
 HADR peer window duration (seconds)  (HADR_PEER_WINDOW) = 0
```

HADR STANDBY 端的数据库通过 PRIMARY 端的数据库备份恢复而来，并且恢复完成后的状态必须是 rollforward-pending 状态。为实现此目的，restore 命令中不能指定选项 WITHOUT ROLLING FORWARD。为了确认 STANDBY 端的状态是否正确，可以通过 db2 get db cfg for <dbname>命令查看，示例如下：

```
$db2 get db cfg for sample|grep  Rollforward
```

```
Rollforward pending                                    = DATABASE
```

在确认上述状态后，就可以启动 HADR。

HADR 的启动有严格的启动顺序：先启动 STANDBY 端，再启动 PRIMARY 端。

首先在 STANDBY 端使用下面的命令启动 HADR：

```
$db2 start hadr on db sample as standby
```

在 STANDBY 端启动成功后，使用监控工具(下面的章节会详细介绍)查看 HADR 的状态，HADR 一般会先处于 local catchup 状态，然后会变为 remote catchup pending 状态。

6.2.5 启动 PRIMARY

在确认 STANDBY 端已经处于正常的状态后，就可以启动 PRIMARY 端。

此时就可以启动 PRIMARY 端的 HADR，使用下面的命令启动 PRIMARY：

```
$db2 start hadr on db sample as primary
```

在 PRIMARY 端启动完成后，使用监控工具查看 HADR 的状态。如果状态是 peer，就表示 HADR 启动成功，db2pd –d sample|grep ' HADR_STATE '示例如下：

```
HADR_STATE = PEER
```

至此，HADR 启动完成。

注意启动 PRIMARY 的时候，PRIMARY 端会等待 STANDBY 建立连接。如果无法与 STANDBY 建立正常的连接，PRIMARY 会启动失败，除非使用 by force 选项强制启动 PRIMARY。因此，正常启动 PRIMARY 的前提条件是 STANDBY 已经正常启动。

6.3 HADR 的维护

本节讨论 HADR 环境下的一些特定维护方式，并且详细介绍 HADR 的各种状态，以及在不同状态下 HADR 发生切换的结果以及后续的处理。

6.3.1 监控 HADR

监控的方法

查看 HADR 的状态和运行情况，主要有两种方式：db2pd 和 snapshot。

首先使用 db2pd 命令查看 HADR 的状态，下面是 PRIMARY 端显示的内容：

```
    db2pd -d sample -hadr
    Database Member 0 -- Database SAMPLE -- Active -- Up 0 days 01:24:03 -- Date
2016-08-25-11.05.04.139848

                            HADR_ROLE = PRIMARY
                          REPLAY_TYPE = PHYSICAL
                        HADR_SYNCMODE = NEARSYNC
                           STANDBY_ID = 1
                        LOG_STREAM_ID = 0
                           HADR_STATE = PEER
                           HADR_FLAGS =
                  PRIMARY_MEMBER_HOST = DRSAMPLEDBA
                     PRIMARY_INSTANCE = db2inst1
                       PRIMARY_MEMBER = 0
                  STANDBY_MEMBER_HOST = DRSAMPLEDBB
                     STANDBY_INSTANCE = db2inst1
                       STANDBY_MEMBER = 0
                  HADR_CONNECT_STATUS = CONNECTED
             HADR_CONNECT_STATUS_TIME = 08/25/2016 10:55:25.243778
(1472093725)
          HEARTBEAT_INTERVAL(seconds) = 15
                     HEARTBEAT_MISSED = 0
                   HEARTBEAT_EXPECTED = 10
                HADR_TIMEOUT(seconds) = 60
        TIME_SINCE_LAST_RECV(seconds) = 5
             PEER_WAIT_LIMIT(seconds) = 5
           LOG_HADR_WAIT_CUR(seconds) = 0.000
    LOG_HADR_WAIT_RECENT_AVG(seconds) = 0.000000
   LOG_HADR_WAIT_ACCUMULATED(seconds) = 0.000
                  LOG_HADR_WAIT_COUNT = 0
SOCK_SEND_BUF_REQUESTED,ACTUAL(bytes) = 1048576000, 1048574992
SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 1048576000, 1048574992
            PRIMARY_LOG_FILE,PAGE,POS = S0631212.LOG, 6, 329325559406386
            STANDBY_LOG_FILE,PAGE,POS = S0631212.LOG, 6, 329325559406386
                  HADR_LOG_GAP(bytes) = 0
     STANDBY_REPLAY_LOG_FILE,PAGE,POS = S0631212.LOG, 6, 329325559406386
       STANDBY_RECV_REPLAY_GAP(bytes) = 67176890
                     PRIMARY_LOG_TIME = 09/19/2016 14:26:27.000000
(1472094304)
                     STANDBY_LOG_TIME = 09/19/2016 14:26:27.000000
(1472093725)
              STANDBY_REPLAY_LOG_TIME = 09/19/2016 14:26:27.000000
(1472093725)
```

```
             STANDBY RECV BUF SIZE(pages) = 1310720
                 STANDBY RECV BUF PERCENT = 0
               STANDBY SPOOL LIMIT(pages) = 52428800
                    STANDBY SPOOL PERCENT = 0
                       STANDBY ERROR TIME = NULL
                     PEER WINDOW(seconds) = 0
                 READS ON STANDBY ENABLED = Y
        STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N
```

下面是 STANDBY 端显示的内容：

```

     Database Member 0 -- Database SAMPLE -- Active Standby -- Up 0 days 00:04:14
-- Date 2016-08-25-11.05.04.376639

                              HADR ROLE = STANDBY
                            REPLAY TYPE = PHYSICAL
                          HADR SYNCMODE = NEARSYNC
                             STANDBY ID = 0
                          LOG STREAM ID = 0
                             HADR STATE = PEER
                             HADR FLAGS =
                    PRIMARY MEMBER HOST = DRSAMPLEDBA
                       PRIMARY INSTANCE = db2inst1
                         PRIMARY MEMBER = 0
                    STANDBY MEMBER HOST = DRSAMPLEDBB
                       STANDBY INSTANCE = db2inst1
                         STANDBY MEMBER = 0
                    HADR CONNECT STATUS = CONNECTED
               HADR CONNECT STATUS TIME = 08/25/2016 10:55:25.243778
(1472093725)
            HEARTBEAT INTERVAL(seconds) = 15
                       HEARTBEAT MISSED = 0
                     HEARTBEAT EXPECTED = 8
                  HADR TIMEOUT(seconds) = 60
          TIME SINCE LAST RECV(seconds) = 5
               PEER WAIT LIMIT(seconds) = 5
             LOG HADR WAIT CUR(seconds) = 0.000
      LOG HADR WAIT RECENT AVG(seconds) = 0.000000
     LOG HADR WAIT ACCUMULATED(seconds) = 0.000
                    LOG HADR WAIT COUNT = 0
    SOCK SEND BUF REQUESTED,ACTUAL(bytes) = 1048576000, 1048574992
    SOCK_RECV_BUF_REQUESTED,ACTUAL(bytes) = 1048576000, 1048574992
```

```
              PRIMARY LOG FILE,PAGE,POS = S0631212.LOG, 6, 329325559406386
              STANDBY LOG FILE,PAGE,POS = S0631212.LOG, 6, 329325559406386
                    HADR LOG GAP(bytes) = 0
        STANDBY REPLAY LOG FILE,PAGE,POS = S0631212.LOG, 6, 329325559406386
          STANDBY RECV REPLAY GAP(bytes) = 67176890
                       PRIMARY LOG TIME = 09/19/2016 14:26:27.000000
(1474266387)
                       STANDBY LOG TIME = 09/19/2016 14:26:27.000000
(1474266387)
                STANDBY REPLAY LOG TIME = 09/19/2016 14:26:27.000000
(1474266387)
            STANDBY RECV BUF SIZE(pages) = 1310720
                STANDBY RECV BUF PERCENT = 0
              STANDBY SPOOL LIMIT(pages) = 52428800
                   STANDBY SPOOL PERCENT = 0
                      STANDBY ERROR TIME = NULL
                    PEER WINDOW(seconds) = 0
                READS ON STANDBY ENABLED = Y
       STANDBY_REPLAY_ONLY_WINDOW_ACTIVE = N
```

此命令是 HADR 重要的监控工具，下面介绍一下相关指标：

- HADR_ROLE：表示当前节点在 HADR 中的角色，分别为 Primary 和 Standby。
- HADR_STATE：表示当前节点所处的 HADR 状态，可能的状态有 Peer、Disconnected、LocalCatchup、RemoteCatchup、RemoteCatchup Pending 和 Disconnected Peer。关于这些状态的含义，我们会在下面的章节中专门讨论。
- HADR_SYNCMODE：表示当前 HADR 使用的日志处理模式，可能的取值有 SYNC、NearSync、Async 和 SuperAsync。
- HeartBeatsMissed：表示在 PRIMARY 和 STANDBY 之间丢失的心跳数量，这项指标对于 HADR 来说很重要。当此值非零的时候，在 SYNC 和 NEARSYNC 模式下，通常意味着 Primary 端的数据库事务会被阻塞。
- HADR_LOG_GAP：此指标表示在 PRIMARY 和 STANDBY 之间的日志 gap，因为网络及处理顺序的原因，STANDBY 端正在 replay 的日志肯定要落后于 PRIMARY 的日志，因此此值一般都是非零的。但此值不能过大，如果过大，就表示 STANDBY 端的 replay 速度慢于 PRIMARY 端创建日志的速度。如果持续下去，势必导致 STANDBY 端的 receive buffer 用尽而导致 PRIMARY 端的事务被阻塞。
- HADR_CONNECT_STATUS：此指标表示 PRIMARY 和 STANDBY 端之间的连接状态，正常的状态是 Connected。如果出现其他的状态，那么应该引起重视。例如，

CONGESTED 状态通常是网络造成 PRIMARY 和 STANDBY 端有严重的延迟。还有另外的一种状态 DISCONNECTED，这表示 PRIMARY 和 STANDBY 端的连接已经断开。

- HADR_CONNECT_STATUS_TIME：表示最近的 ConnectStatus 状态为 Connected 的时间。
- HADR_TIMEOUT：表示 HADR 设置参数 hadr_timeout 的值。
- PRIMARY_MEMBER_HOST：表示 HADR 主机所处的主机名。
- STANDBY_MEMBER_HOST：表示 HADR 备机所处的主机名。
- PRIMARY_INSTANCE：表示 HADR 主机的 DB2 实例名。
- STANDBY_INSTANCE：表示 HADR 备机的 DB2 实例名。
- PRIMARY_LOG_FILE,PAGE,POS：表示 PRIMARY 端活动的日志文件名称、数据页号、日志流的偏移量(字节)。
- STANDBY_LOG_FILE,PAGE,POS：表示 STANDBY 端活动的日志文件名称、数据页号、日志流的偏移量(字节)。
- STANDBY_RECV_BUF_PERCENT：表示 STANDBY 端用于缓冲日志的缓冲区的使用率。
- STANDBY_SPOOL_PERCENT：表示 STANDBY 端 SPOOL 的使用率。当 STANDBY_RECV_BUF_PERCENT 用满时，会把 LOG 移动到 SPOOL 里，当 STANDBY_RECV_BUF_PERCENT 和 STANDBY_SPOOL_PERCENT 都接近 100 时，如果当前同步模式是 NEARSYNC，会阻塞主机的写交易。

监控的最佳实践

在日常运行过程中需要时刻关注 HADR 的状态，尤其是丢失心跳和非 peer 状态。

心跳的丢失会导致 PRIMARY 端数据库被阻塞(在 SYNC 和 NEARSYNC 模式下)，因此 HEARTBEAT_MISSED 这个指标非常重要，需要时刻关注。

对非 peer 状态的监控也非常重要，当 HADR 处于非 peer 状态时，表示 HADR 的 STANDBY 端数据库已经与 PRIMARY 端数据库处于不一致的状态，此时 PRIMARY 端的数据库已经失去了 STANDBY 端高可用保护，因此这种情况下需要尽快找到问题的原因并解决，以使 STANDBY 端数据尽快一致，使状态恢复至 peer 状态。

通常情况下，需要在生产的 HADR 环境中使用自动化监控工具时刻监控 HADR 的状态，当出现状态异常时迅速告警，使运维人员能够及时发现问题并处理。

6.3.2　HADR 的切换方式

HADR 的高可用能力是通过将 STANDBY 角色切换为 PRIMARY 角色实现的。当 HADR 的状态处于 peer 状态时，就可以在 STANDBY 端使用 takeover 命令来实现上述切换，例如：

```
$db2 takeover hadr on sample
DB20000I  The TAKEOVER HADR ON DATABASE command completed successfully.
```

在上面的命令执行成功后，我们将会发现原来的角色发生了互换，原来的 PRIMARY 切换为 STANDBY，而原来的 STANDBY 切换为 PRIMARY。

注意上面的命令只能在 STANDBY 端执行切换，如果在 PRIMARY 端执行上面的命令，就会收到下面的报错：

```
$db2 takeover hadr on db sample
SQL1770N  Takeover HADR cannot complete. Reason code = "4".
```

除了通过上述命令实现正常切换之外，还可以使用强制切换选项来完成角色的切换，命令如下：

```
$db2 takeover hadr on db sample by force
DB20000I  The TAKEOVER HADR ON DATABASE command completed successfully.
```

此命令会将原来的 STANDBY 端切换为 PRIMARY，而原来的 PRIMARY 并不能切换为 STANDBY，而是继续保留为 PRIMARY。这样就很容易造成两端都是 PRIMARY 的情况，从而造成数据的不一致，因此应该慎用此命令。

尤其是在 HADR 处于非 peer 状态时，如果使用此命令强制切换，很有可能会造成数据的丢失。

6.3.3　切换后对应用产生的影响

在切换后，HADR 的角色会发生互换，旧的 STANDBY 成为新的 PRIMARY，旧的 PRIMARY 则成为新的 STANDBY。那么应用程序在访问数据库的时候，如果还指向旧的 PRIMARY 端，将无法正常访问数据库，因为此时它已经成为 STANDBY 端，而 STANDBY 端无法提供正常的数据库访问。此时就需要应用程序的数据库连接信息重新指向新的 PRIMARY 端，这样才能继续访问数据库。

另外，DB2 本身还提供了 reroute 功能，通过设置可以使数据库的客户端应用自动 reroute 到新的地址，此功能正好可以与 HADR 功能配合使用，当旧的 STANDBY 切换为新的 PRIMARY 后，客户端会自动重新找到新的 PRIMARY 数据库，如图 6-3 所示。

使用如下命令配置相关参数：

```
db2 update alternate server for database sample using hostname HADRDBB port DBBPORT
```

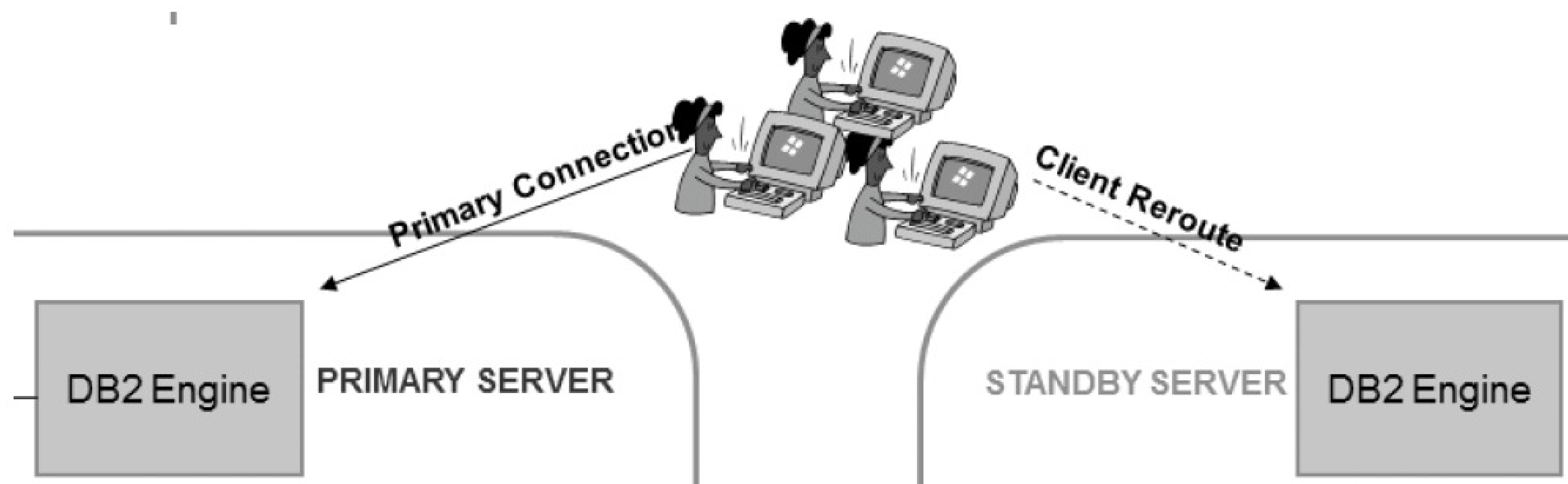


图 6-3　客户端路由

这项技术支持的客户端类型可以是 Java 客户端(JDBC、SQLJ 或 pureQuery)，也可以是非 Java 客户端(ODBC、CLI、.NET、OLE DB、PHP、Ruby 或 embedded SQL)。

对于 Java 客户端，使用的驱动类型需要为 type 4，并且版本需要为 3.58、4.8 或更新版本。

当客户端接收到通信错误(sqlcode -30081)或 sqlcode -1224 错误码时，客户端将会根据数据库的设置执行自动路由，使用新的主机和端口访问新的数据库。

6.3.4　HADR 状态

HADR 可以处于很多不同的状态，下面分别看看这些状态的意义和需要的处理方法。

- Peer：表示 HADR 的两端处于对等的同步状态，两端数据库的数据都是一致的。当 PRIMARY 端在此状态下发生故障时，STANDBY 完全可以在不丢失任何数据的情况下切换为 PRIMARY，从而替代原来 PRIMARY 的职能。
- Disconnected：表示 PRIMARY 与 STANDBY 端失去了连接，此状态通常在 PRIMARY 端显示。出现这种状态的原因可能是网络真的发生了中断、可能是 STANDBY 端的数据库已经停止等原因。这种状态下，HADR 是无法提供高可用保护的。
- LocalCatchup：表示 STANDBY 端正在 replay 本地日志，此状态通常表示 STANDBY 处于启动的第一个阶段，或是从中断的状态重新恢复。处于这个状态，通常意味着 HADR 的 STANDBY 正在从本地获取日志并 replay 这些日志。如果这些日志较

多，那么此状态的持续时间会较长；而如果这些日志很少或没有，那么此状态的时间会很短暂。在这种状态下，HADR 也是无法提供数据一致的高可用保护的。

- RemoteCatchup：表示 STANDBY 端正在从 PRIMARY 获取日志，并且异步 replay 这些日志。在 SYNC 和 NEARSYNC 模式下，此状态表示 STANDBY 端的数据与 PRIMARY 端是不一致的，是无法提供数据一致的高可用保护的。
- RemoteCatchupPending：表示 STANDBY 端正在等待从 PRIMARY 端获取日志。如果 STANDBY 持续处于此状态，就表示 STANDBY 未能与 PRIMARY 建立连接，否则此状态只会是短暂的过程。
- DisconnectedPeer：只有在设置 hadr_peer_window 的值大于零的情况下才可能出现，此状态表示 PRIMARY 端与 STANDBY 端的连接已经中断，但是时间还没超出参数 hadr_peer_window 设置的限制。在此状态下，通常意味着 PRIMARY 和 STANDBY 端的数据是一致的，此时发生 takeover 后，不会发生数据的丢失。

6.3.5　HADR 异常状态的处理

一般情况下，正常运行的 HADR 应该处于 peer 状态。当处于非 peer 状态时，就需要我们采取必要的措施进行处理，从而恢复 peer 状态。或者在非 peer 状态下，PRIMARY 端又因为异常而导致无法正常运行时，需要采用相应的措施才能确保数据不会丢失。

6.4　HADR 性能调优

6.4.1　接收缓冲

日志从 PRIMARY 发送到 STANDBY 后会被存放在 HADR 专用的内存缓冲池中，HADR REPLAY 线程在需要 replay 日志的时候才会从缓冲池中读取日志内容并执行 replay。

在 STANDBY 端，缓冲池的大小由注册变量 DB2_HADR_BUF_SIZE 定义，此缓冲池的默认设置为两倍的 logbufsz 设置，比如数据库的参数 logbufsz 设置为 512，对于 STANDBY 的默认缓冲池大小为 2*512*4KB=4096KB。

在某些场景下，缓冲池的大小对于 HADR 的性能有显著影响。

当 STANDBY replay 日志的速度慢于 PRIMARY 端生成日志的速度时，缓冲池的使用率将会越来越高。在 HADR 的同步模式设置为 SYNC 或 NEARSYNC 时，缓冲池的使用率达到 100%会导致 PRIMARY 的性能下降，从而保证 STANDBY 端的缓冲日志不会超过缓冲池的大小，也就是 PRIMARY 的性能会与 STANDBY 端 replay 日志的性能一致。

因此，为了避免影响 PRIMARY 端的性能，应该保证缓冲池的大小满足绝大多数情况

下系统出现抖动而不会影响 PRIMARY 的性能，比如可以将缓冲池设置为 1GB 或 2GB。而且应该时刻监控缓冲池使用率的变化，当使用率达到 50%或其他更高的指标时，就应该决定 PRIMARY 端的性能是否很重要。如果无论如何也不能影响 PRIMARY 的性能，就应该选择停止 STANDBY 的 HADR 线程以避免影响 PRIMARY 的性能。

6.4.2 网络相关

DB2 提供了两个用于设置网络缓冲的参数：DB2_HADR_SOSNDBUF 和 DB2_HADR_SORCVBUF。这两个参数用于指定进行 HADR 通信时在 TCP 上发送数据的缓冲池和接收数据的缓冲池的大小。一般可以适当地调整这两个值，比操作系统提供的值稍大即可。

另外，IBM 提供了测试程序 simhadr，用于测试 HADR 两端的通信是否存在性能瓶颈，可以使用这个程序测试得出性能最好的设置。

6.4.3 内部参数

DB2 还通过注册变量配置参数 DB2BPVARS 提供了几个内部参数，这些参数用于设置 STANDBY 端在 replay 日志时的并发情况，设置信息如下：

```
DB2BPVARS=/db2/db2hadr/sqllib/DB2BPVARS.cfg
$cat /db2/db2hadr/sqllib/DB2BPVARS.cfg
PREC_NUM_AGENTS=32
PREC_NUM_QSETSIZE=32
PREC_NUM_QSETS=32
PREC_NUM_WOCB=2048
```

上述参数值的最佳设置与 OS 环境关联很紧密，比如物理 CPU 的数量，所以可以在自己的环境中采用不同的组合设置进行测试，进而找到最适合自身环境的设置。

6.4.4 表和表空间的调整

HADR 的 STANDBY 端在 replay 日志的时候，大多数情况是可以并行执行的，但有些个别情况就只能并行执行，例如对表空间级别的 extent 映射信息的修改，这些修改在 STANDBY 端只能串行地执行，这种情况就有可能会影响 STANDBY 的效率，从而造成与 PRIMARY 之间的 gap 越来越大。为了最大程度地避免这一问题，可以采取两个措施：

- 针对压力大的表所在的表空间，增加 extentsize 和 pagesize 的大小。
- 将压力较大的表放置在单独的表空间中，避免与其他压力大的表位于同一表空间。

实践证明，上述方法对提高 HADR STANDBY 的 replay 性能很有帮助；但是一定要通过正确方法定位到正确的表。

6.5　HADR 高可用案例分享

在采用 HADR 技术作为数据库的高可用方案后，我们还面临着如下问题，就是如何将数据库高可用与其他高可用方案进行结合，如图 6-4 所示。因为 HADR 技术并非万能，有很多高可用需求是无法直接满足的。例如在生产环境中，当主机宕机后，希望数据库能够自动完成快速切换以提高最大可用性，而 HADR 技术本身无法单独完成上述需求，因为 HADR 不能判断主机的宕机故障，而且也无法自动完成 takeover 切换操作，需要手工执行才可以，这就大大限制了 HADR 技术应用的场景。

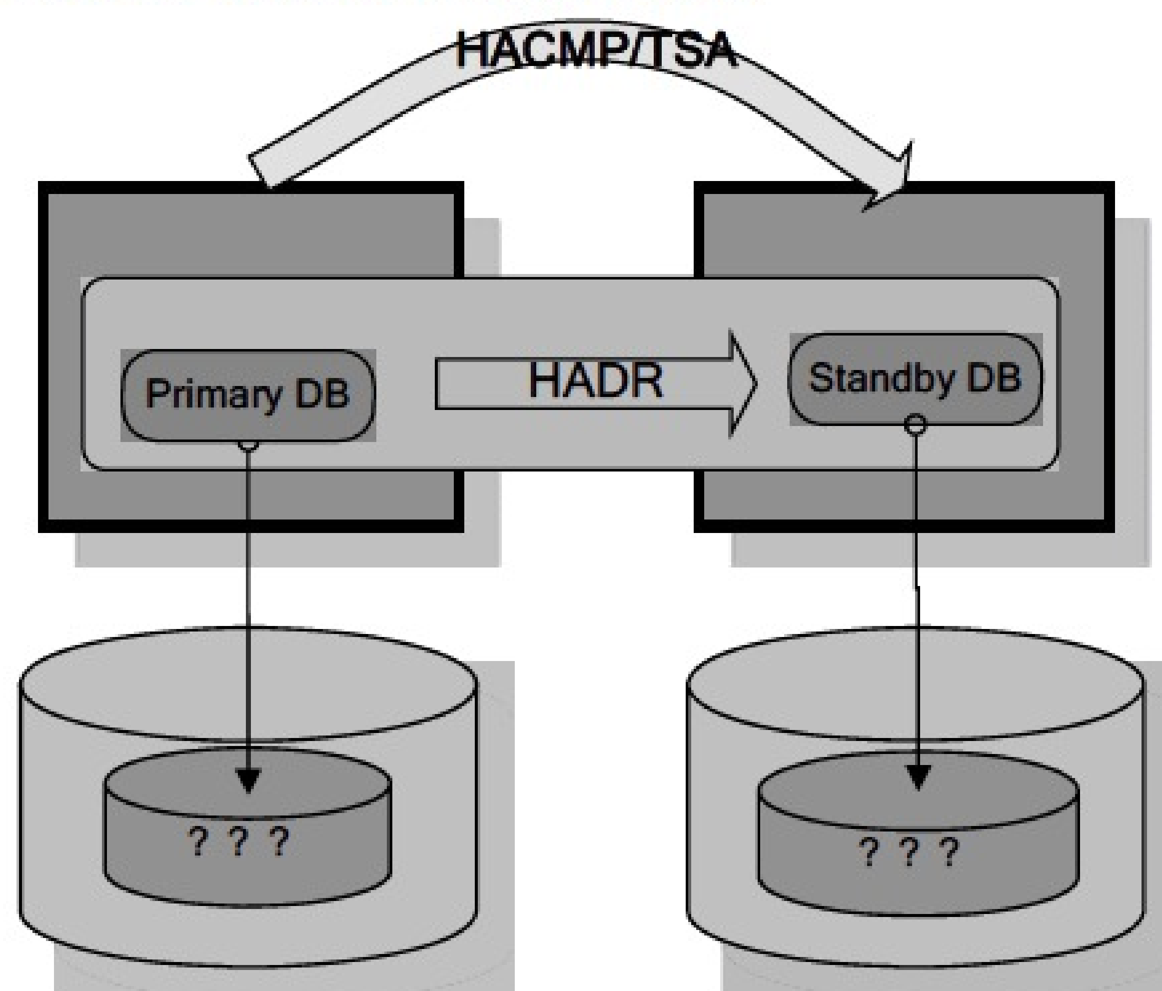


图 6-4　HADR 结合其他高可用软件的架构

但是当我们将 HADR 技术与成熟的主机高可用产品结合在一起的时候，上面的问题就会迎刃而解。下面简单介绍两种在 IBM 产品中常见的主机高可用产品与 HADR 结合的案例。

6.5.1　HADR 结合 PowerHA

PowerHA 介绍

PowerHA(以前称为 HACMP)是 IBM 在基于 Power 系列芯片的平台上提供的高可用产品，目前可以支持 AIX 和 Linux 操作系统，支持三类主机层次故障的自动诊断和处理：网卡故障、网络故障、主机故障。通过灵活的自定义功能，还可以扩展其他类型故障的诊断和处理，比如应用程序或业务的故障诊断和处理等。

PowerHA 实现高可用功能的一些前提条件为：

- 冗余服务器
- 冗余网络
- 冗余网络适配器

- 监视
- 故障检测
- 故障诊断
- 自动化的故障转移
- 自动化的重新集成

具体的单点故障列表和 PowerHA 的处理办法如表 6-1 所示。

表 6-1　PowerHA 消除单点故障的方法

单点故障对象	消除单点故障的方法
节点(服务器)	多个节点
电源	多个电路和/或电源
网络适配器	冗余网络适配器
网络	用于连接节点的多个网络
TCP/IP 子系统	非 IP 网络以便为 TCP/IP 提供后备
磁盘适配器	冗余磁盘适配器
磁盘	冗余硬件和磁盘镜像或 RAID 技术
应用程序	配置应用程序监视和备份节点以获取应用程序引擎和数据

PowerHA 给 HADR 提供哪些增强

通过上面的介绍可以得知，PowerHA 可以诊断多种故障并可以自动完成处理，处理的内容可以通过配置资源组的方式来实现，PowerHA 一般可以把下面的内容配置到资源组中：IP 地址、卷组、文件系统、NFS、应用的启停脚本等资源。

但对于 HADR 环境而言，由于用来保证 DB2 启动运行的所有资源都是本地的，因此一般只需要在 PowerHA 的资源组中配置对外提供服务的服务 IP 和数据库的启停脚本即可。

因此，PowerHA 将会在 PRIMARY 端发生异常的情况下，完成对 HADR 的切换，并自动将对外服务的 IP 地址切换到 STANDBY 端，这样客户端只会发生很短暂的中断，就可以继续访问数据库了。

PowerHA 在 HADR 环境下的设置

配置 PowerHA 的步骤和过程如下：

- 操作系统准备：安装 PowerHA 软件，配置网络，配置/etc/hosts 文件等
- 数据库准备：安装 DB2，创建实例，创建数据库，配置 HADR
- 创建 PowerHA 集群

- 创建 PowerHA 节点
- 创建 PowerHA 网络
- 创建 PowerHA 接口
- 创建 PowerHA 资源：应用服务器和服务 IP

下面详细介绍一下应用服务器和服务 IP 的配置，其他步骤在此不再详细说明，请另外参阅其他资料。

- 应用服务器的配置

对于 HADR 数据库的启停脚本，DB2 官方提供了一套启停脚本。一般情况下，这套脚本还是适用的，路径和名称为：

- 启动脚本：$INSTANCE_DIR/sqllib/samples/hacmp/rc.hadr.start
- 停止脚本：$INSTANCE_DIR/sqllib/samples/hacmp/rc.hadr.stop
- 监控脚本：$INSTANCE_DIR/sqllib/samples/hacmp/rc.hadr.monitor

使用上面的脚本配置 PowerHA 的应用服务器即可，如下所示：

```
    smitty hacmp→Extended Configuration→Extended Resource Configuration→
HACMP Extended Resources Configuration→Configure HACMP Application Servers→
Configure HACMP Application Servers→Add an Application Servers
```

```
                          Add Application Server

    Type or select values in entry fields.
    Press Enter AFTER making all desired changes.

                                         [Entry Fields]
    * Server Name               [db2hadr app]
    * Start Script              [/db2/db2hadr/sqllib/samples/hacmp/rc.hadr.start]
    * Stop Script               [/db2/db2hadr/sqllib/samples/hacmp/rc.hadr.stop]
      Application Monitor Name(s)                  +
```

- 服务 IP 的配置

PowerHA 中服务 IP 地址的设置相对简单，可使用下面的路径来配置：

```
    smitty hacmp→Extended Configuration→Extended Resource Configuration→
HACMP Extended Resources Configuration→ Configure HACMP Service IP Labels/
Addresses→ Add a Service IP Label/Address
```

```
     Select a Service IP Label/Address type                              |
```

```
| Move cursor to desired item and press Enter.                    |
|   Configurable on Multiple Nodes                                |
|   Bound to a Single Node
```

```
Add a Service IP Label/Address configurable on Multiple Nodes (extended)

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                  [Entry Fields]
* IP Label/Address                              HADR SVC                +
  Netmask(IPv4)/Prefix Length(IPv6)                 [255.255.255.0]
* Network Name                                     net ether 01
  Alternate HW Address to accompany IP Label/Address []
```

在上面的菜单中选择并填写好相应的 IP 地址即可。

将上面的两项资源配置到已经创建好的资源组中即可。

```
Service IP Labels/Addresses                        [HADR SVC]           +
  Application Servers                              [db2hadr_app]
```

PowerHA 结合 HADR 的简单维护

PowerHA 脚本对 HADR 的处理方式如表 6-2 所示。

表 6-2　HACMP 脚本对 HADR 的处理

脚本名称	判断	当前实例的 HADR 状态	处理办法
rc.hadr.start	"$start_as_standby" = "S"	所有状态	启动为 STANDBY
	"$start_as_standby" != "S"	当前数据库是 PRIMARY，并且状态为 peer	无
		当前数据库是 PRIMARY，状态是非 peer	无
		当前数据库是 STANDBY，状态是 peer	执行 takeover (no force)
		当前数据库是 STANDBY，状态是非 peer	执行 takeover by force
		如果不属于以上状态，将判断 HADR 为 standard 状态	将当前数据库启动为 PRIMARY
rc.hadr.stop		peer 状态	db2_kill
		非 peer 状态	无

PowerHA 在切换过程中调用应用脚本的顺序为：先在 PRIMARY 端调用应用的停止脚本，再在 STANDBY 端调用启动脚本。

- 主切备

切换可以通过移动资源组的形式完成，smitty 菜单的路径为：

smitty hacmp→System Management(C-SPOC)→Resource Group and Applications→Move a Resource Group to Another Node/Site→ Move Resource Groups to Another Node

通过上面的表 6-2，我们可以看到在正常的 PRIMARY 端执行停止脚本 rc.hadr.stop 时，会执行 db2_kill 命令，这样 DB2 实例将会被立刻终止。接着，PowerHA 会在 STANDBY 端执行启动脚本 rc.hadr.start，而此时 STANDBY 端将会处于 remote catchup pending 状态，根据表 6-2 中列出的处理方法，启动脚本将会执行 takeover by force 命令，此时 STANDBY 端将会被强制切换至 PRIMARY 角色。

- 切换后的恢复

在主切备执行后，我们会发现虽然旧的 STANDBY 被切换为 PRIMARY 以继续提供数据库服务，但是原来的 PRIMARY 端的数据库实例仍然是停止的，此时新的 PRIMARY 没有 HADR 的高可用保护，因此在完成切换后，必须尽快恢复 HADR 的正常状态。

默认情况下，PowerHA 没有提供完成上述功能的方法，因此需要我们手工来恢复 HADR。此时需要将原有的 PRIMARY 端启动并转换为 STANDBY 的角色，步骤是：

(1) 启动数据库实例：

```
$db2start
```

(2) 启动数据库为 STANDBY 角色：

```
$db2 start hadr on db sample as standby
```

总结

PowerHA 与 HADR 的结合为我们提供了一种非常简单和实用的自动化高可用方案，通过 PowerHA 自动处理故障的能力实现故障处理的自动化，通过 HADR 提供的数据库高可用能力，实现数据库服务的高可用，与传统的共享存储的高可用方案比较起来，这种方案具有可用性更高、故障处理的速度更快的优点。

6.5.2 HADR 结合 TSA

TSA 介绍

TSA(Tivoli System Automation，另外的名称是 Tivoli SA MP)是 IBM 提供的另一款高可用软件，基本原理也是通过各种冗余的资源、通过自动化的检测故障实现服务的高可用性。

TSA 与 PowerHA 在使用上的最大不同是使用的方式区别很大，TSA 只能通过配置文件和命令来完成配置、管理和监控，而 PowerHA 提供了比较好用的菜单。

另外，在 DB2 V9.5 及以后版本的 ESE 版本中，TSA 已经随 DB2 的安装介质同时发布了，在安装 DB2 数据库 ESE 版本的时候同时会安装 TSA 软件。在 AIX 操作系统中，可以通过下面的命令确认是否安装了 TSA：

```
    #lslpp -l|grep  "^  sam"
      sam.adapter                 3.2.2.1  COMMITTED  SAM adapter for end-to-end
      sam.core.rte                3.2.7.1  COMMITTED  SA CHARM Runtime Commands
      sam.msg.DE DE.core           3.2.2.0  COMMITTED  SAM Msgs - German (UTF)
      sam.msg.ES ES.core           3.2.2.0  COMMITTED  SAM Msgs - Spanish (UTF)
      sam.msg.FR FR.core           3.2.2.0  COMMITTED  SAM Msgs - French (UTF)
      sam.msg.IT IT.core           3.2.2.0  COMMITTED  SAM Msgs - Italian (UTF)
      sam.msg.JA JP.core           3.2.2.0  COMMITTED  SAM Msgs - Japanese (UTF)
      sam.msg.Ja JP.core           3.2.2.0  COMMITTED  SAM Msgs - Japanese (IBM-943
      sam.msg.KO KR.core           3.2.2.0  COMMITTED  SAM Msgs - Korean (UTF)
      sam.msg.PT BR.core           3.2.2.0  COMMITTED  SAM Msgs - Brazilian
      sam.msg.ZH CN.core           3.2.2.0  COMMITTED  SAM Msgs - Simplified
Chinese
      sam.msg.ZH TW.core           3.2.2.0  COMMITTED  SAM Msgs - Traditional
Chinese
      sam.msg.Zh CN.core           3.2.2.0  COMMITTED  SAM Msgs - Simplified
Chinese
      sam.msg.Zh TW.core           3.2.2.0  COMMITTED  SAM Msgs - Traditional
Chinese
      sam.msg.de DE.core           3.2.2.0  COMMITTED  SAM Msgs - German
(ISO8859-1)
      sam.msg.es ES.core           3.2.2.0  COMMITTED  SAM Msgs - Spanish
(ISO8859-1)
      sam.msg.fr FR.core           3.2.2.0  COMMITTED  SAM Msgs - French
(ISO8859-1)
      sam.msg.it IT.core           3.2.2.0  COMMITTED  SAM Msgs - Italian
(ISO8859-1)
      sam.msg.ja JP.core           3.2.2.0  COMMITTED  SAM Msgs - Japanese
      sam.msg.ko KR.core           3.2.2.0  COMMITTED  SAM Msgs - Korean
(IBM-eucKR)
```

```
      sam.msg.pt BR.core        3.2.2.0  COMMITTED  SAM Msgs - Brazilian
      sam.msg.zh TW.core        3.2.2.0  COMMITTED  SAM Msgs - Traditional
Chinese
      sam.policies.one         3.2.2.1  COMMITTED  IBM Tivoli System
Automation
      sam.policies.two         3.2.2.1  COMMITTED  IBM Tivoli System
Automation
      sam.sappolicy            3.2.2.1  COMMITTED  IBM Tivoli System
Automation
      sam.adapter              3.2.2.1  COMMITTED  SAM adapter for end-to-end
      sam.core.rte             3.2.7.1  COMMITTED  SA CHARM Runtime Commands
      sam.policies.one         3.2.2.1  COMMITTED  IBM Tivoli System
Automation
      sam.policies.two         3.2.2.1  COMMITTED  IBM Tivoli System
Automation
      sam.sappolicy            3.2.2.1  COMMITTED  IBM Tivoli System
Automation
```

上面的输出表示已经安装了 TSA 软件。

TSA 给 HADR 提供哪些增强

TSA 为了更好地支持 DB2 产品，提供了专门的配置工具 db2haicu，在安装了 DB2 V9.7 后，可以在 sqllib/bin 目录中找到此工具。db2haicu 工具的使用有两种模式：一种是交互模式；另一种是 XML 模式。交互模式通过 db2haicu 工具的提示填入必要的信息以完成配置过程，XML 模式通过读取填写好的符合规范的 XML 文件来完成配置过程。

TSA 能自动处理的故障有：

- 节点故障
- 网络故障
- 磁盘故障
- 数据库进程故障

一般出现上述故障时，TSA 会自动识别故障的类型并采取预定义好的方法进行处理。

TSA 在 HADR 环境下的设置

通过 TSA 的交互模式配置 DB2 的基本过程如下：

- 创建集群域
- 仲裁(Quorum)设置

- 网络设置
- 集群管理器选择
- 故障恢复策略
- 设置服务 IP 地址
- 创建数据库及相关数据库对象

通过 TSA 创建上面的内容后，TSA 将会自动完成上述资源的设置，并能够通过 TSA 的 lssam 命令查看相关的资源和状态。

TSA 的设置和管理都可以通过 db2haicu 完成，详细的使用方法可以参阅相关资料，下面只显示已经存在的 TSA 可以使用的管理功能。

在使用 db2haicu 命令后，我们可以看到下面的提示，同时可以选择使用的功能：

```
    db2haicu
    Welcome to the DB2 High Availability Instance Configuration Utility
(db2haicu).
    You can find detailed diagnostic information in the DB2 server diagnostic
log file called
    db2diag.log. Also, you can use the utility called db2pd to query the status
of the cluster
    domains you create.
    For more information about configuring your clustered environment using
db2haicu, see the
    topic called 'DB2 High Availability Instance Configuration Utility
(db2haicu)' in the DB2
    Information Center.
    db2haicu determined the current DB2 database manager instance is db2inst1.
The cluster
    configuration that follows will apply to this instance.
    db2haicu is collecting information about your current setup. This step may
take some time as
    db2haicu will need to activate all databases for the instance to discover
all paths ...
    When you use db2haicu to configure your clustered environment, you create
cluster domains.
    For more information, see the topic 'Creating a cluster domain with db2haicu'
in the DB2
    Information Center. db2haicu is searching the current machine for an existing
active cluster
    domain ...
    db2haicu found a cluster domain called HA domain on this machine. The cluster
configuration
```

```
that follows will apply to this domain.
Select an administrative task by number from the list below:
1.Add or remove cluster nodes.
2.Add or remove a network interface.
3.Add or remove a highly available database.
4.Add or remove a mount point.
5.Add or remove an IP address.
6.Add or remove a non-critical path.
7.Move DB2 database partitions and HADR databases for scheduled maintenance.
8.Change failover policy for this instance.
9.Create a new quorum device for the domain.
10.Destroy the domain.
11.Exit.
```

总结

TSA 提供了另一种可选的自动处理异常的高可用方案，由于 TSA 已经随着 DB2 的介质一起发布，因此 IBM 在一些特定场景下提供了更简单、更自动化的配置方法，例如 SAP 的 DB2 介质中就包括了一套自动化的配置脚本，只需要回答几个设置项就可以自动完成一套高可用环境的配置。

第 7 章 DB2 集群与同城双活

DB2 在 V9.7 的同时期发行了特殊的集群版本 V9.8，也就是 pureScale 集群。从 V10 开始，pureScale 作为 DB2 的可选功能组件，和 DPF 一样，包含在企业版和高级企业版的软件许可中。本章所说的 DB2 集群就是专指 pureScale 特性。

DB2 pureScale 集群就像 Oracle RAC 技术，是面向 OLTP 的高可用、可横向扩展的集群解决方案。DB2 pureScale 集群是区别于传统主备 HA 模式的本地高可用解决方案。在此基础上，DB2 pureScale 集群也可以扩展到同城双数据中心，也就是 GDPC 同城双活容灾解决方案。DB2 pureScale 集群还可以和 HADR 特性相结合，提供异地灾备解决方案。这一章将详细介绍这些解决方案的搭建和维护。

本章主要讲解如下内容：

- DB2 集群介绍
- DB2 集群搭建
- DB2 集群维护
- DB2 集群设计调优
- 同城双活集群介绍
- DB2 集群异地容灾

7.1 DB2 集群介绍

DB2 pureScale 是同时兼备高扩展性和高可用性的数据库集群，同时对应用是透明的。无论连接到集群内哪个成员节点，连接到的都是同一个数据库。图 7-1 总结了 DB2 集群的技术架构。

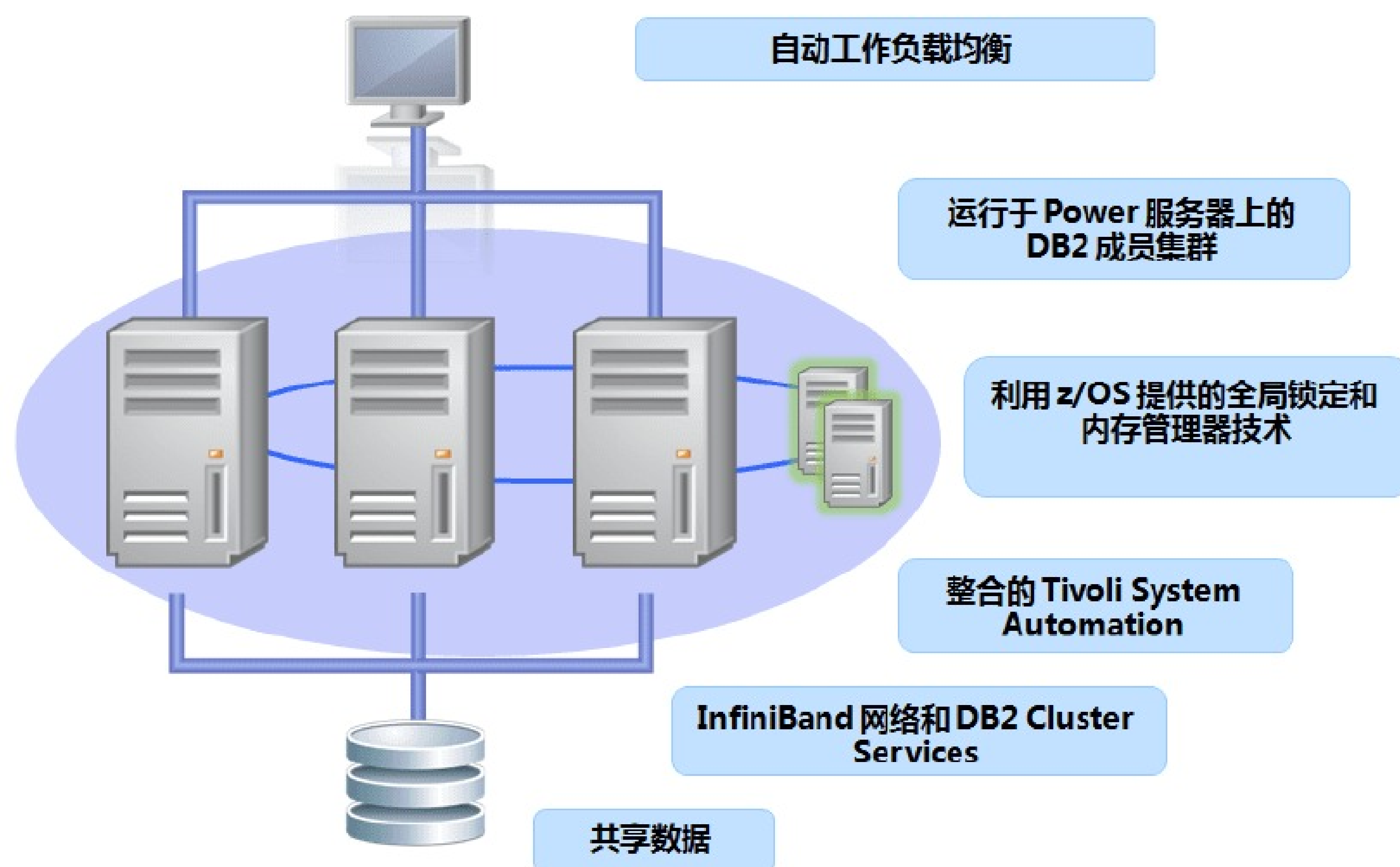


图 7-1 DB2 pureScale 集群架构

首先，DB2 集群是共享存储架构，每个数据库处理节点(成员)都可以直接操作磁盘数据。共享存储使用的是 IBM GPFS 集群文件系统。这个集群文件系统是整个 DB2 集群的最底层基础，被 DB2 集群所管理，而整个 DB2 集群的管理软件核心是 Tivoli System Automation(TSA)集群软件。TSA 负责监控整个集群里面资源的状态，包括存储、网络、进程等，并且基于这些资源的状态自动化相应的行为。例如 TSA 监控到某个节点的网卡宕机，就会自动关闭当前节点的资源，并漂移到其他健康节点。

在共享存储的基础上，集群内部的通信对响应时间要求很高。DB2 pureScale 集群支持 RDMA 协议和 TCP/IP 协议。RDMA 协议相对速度更快，资源消耗更小，在高性能运算中建议使用。而 TCP/IP 协议部署成本低，适合于高可用场景。为了协调各节点一起服务同一个数据库，在 DB2 集群内部引入了 CF 功能，这个功能推荐作为单独的节点。CF 节点主要提供全局锁定和内存管理器技术，继承于 IBM z/OS 系统。现在 DB2 集群支持 Power 服务器和 x86 服务器两种开放式平台。

DB2 集群数据库处理节点多，对于上层应用来说，客户端可选的连接方式也比较齐全。

主要运用的客户端连接方式有两种：自动工作负载均衡和客户端偏好设置。 如果使用自动工作负载均衡，那么数据库服务器会不断反馈给客户端当前的机器节点负载列表，客户端会基于此列表分发事务；而如果选择客户端偏好设置，那么客户端会一直连接首选的数据库成员节点，只有在这个偏好的节点连接不上时，才会连接设置的下一个节点。这两种方式各有利弊。

7.2　DB2 集群的搭建

这一节主要关注如何设计和搭建一套 pureScale 集群。这也是一个实际案例，在某金融行业的一个项目中需要使用一套 DB2 pureScale 集群来提高可用性。这个项目使用了 IBM 的 Power 机器，集群内部的通信网络采用了基于万兆以太网的 ROCE 网络。

7.2.1　系统物理架构

在本项目的 DB2 pureScale 集群中我们要使用三种类型的网络，分别是普通千兆以太网络、RoCE 万兆以太网络和存储 SAN 光纤网络，整个系统的物理连接图如图 7-2 所示。

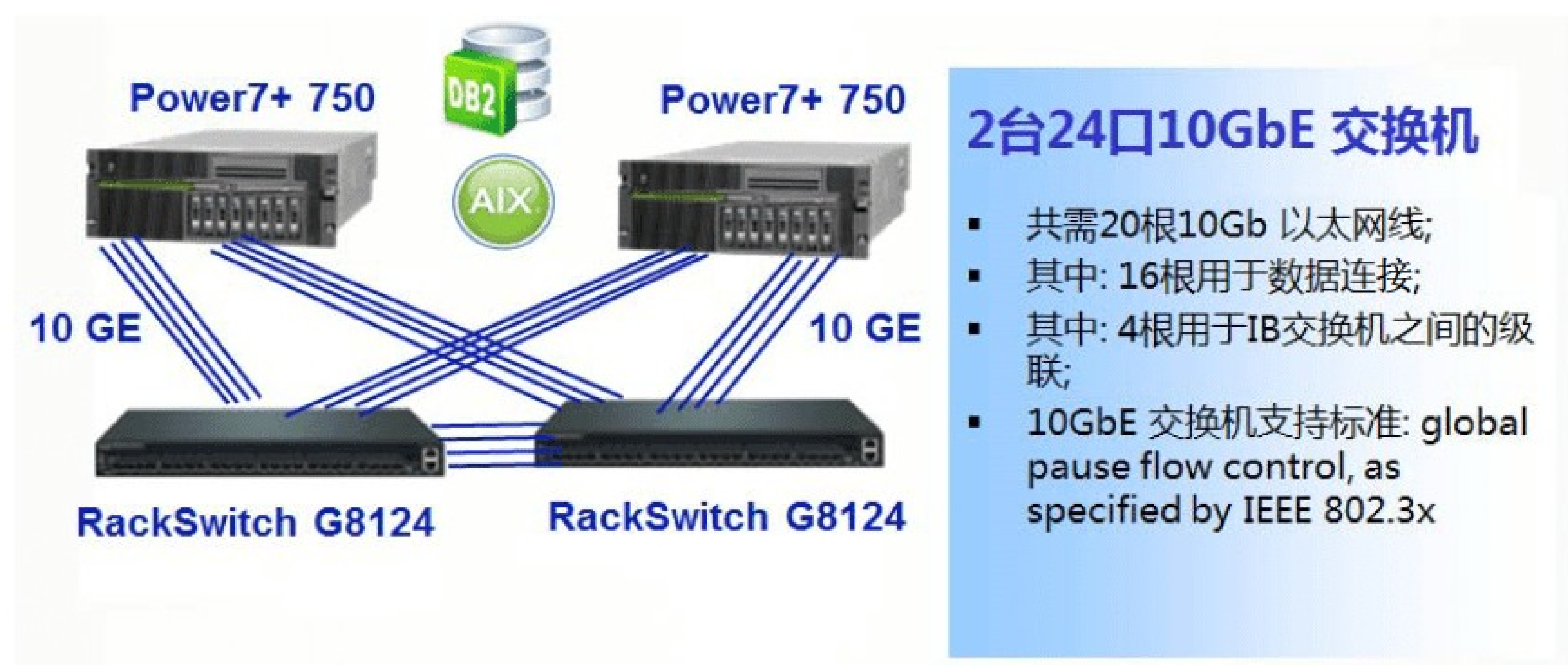


图 7-2　系统物理架构

- 每个 Power 750 配置四块双口 IBM RoCE 万兆网卡，分别连接到两台 IBM G8124 万兆以太网交换机上。
- 两台 IBM G8124 万兆交换机通过级联提供交换机的高可用性。
- 每个物理机器分出两个 LPAR，CF/Member 部署在一台物理机器上。

实际环境里面的网络接线如图 7-3 所示，机器上的网卡接口要接到不同的网络交换机上，这样在交换机发生单点故障的时候，网络不受影响。交换机之间的互联线要保证冗余和带宽。

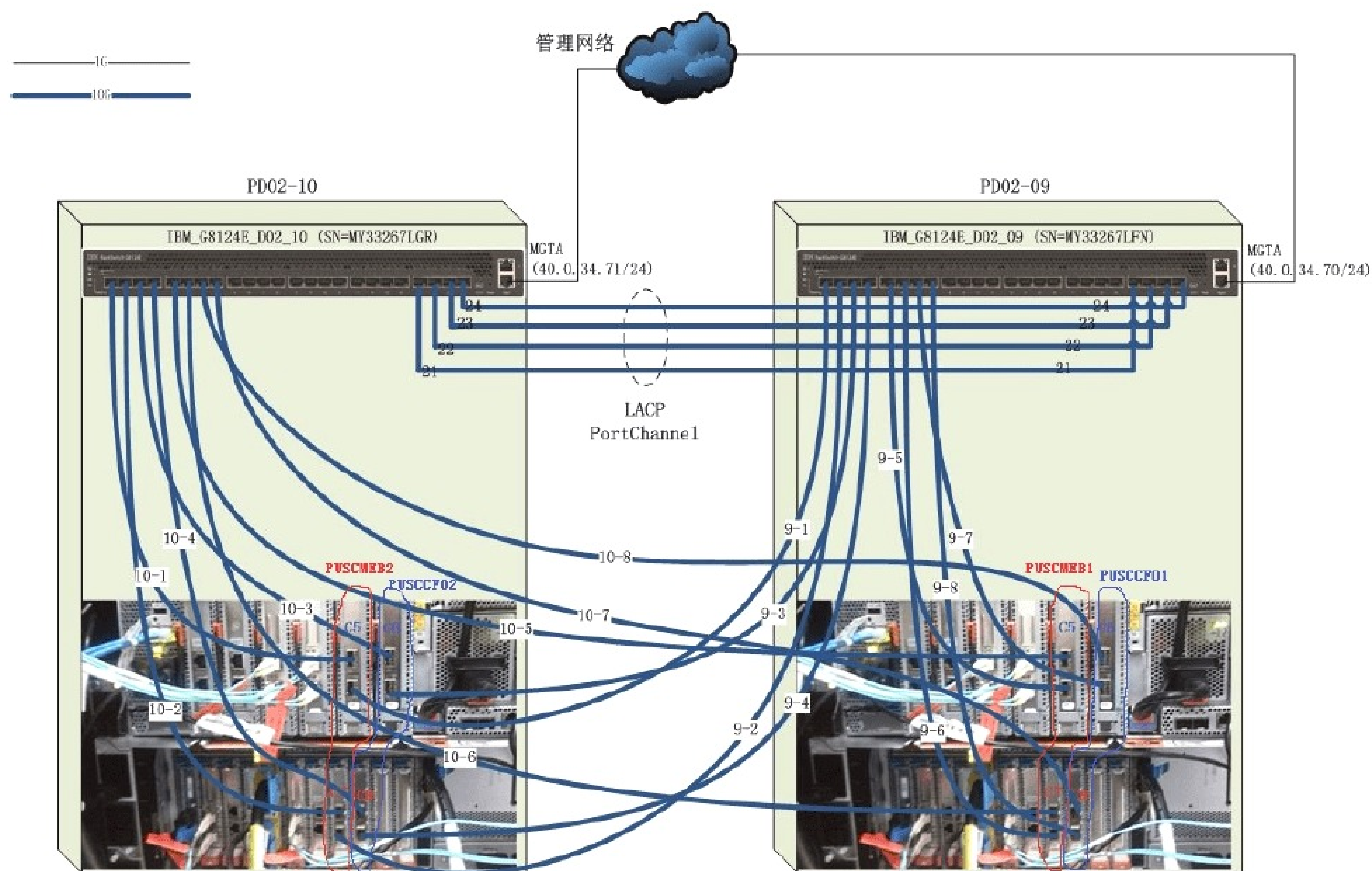


图 7-3　系统网络规划

这个环境里面的 SAN 网络没有画图展示。每个机器通过 SAN 网络直连共享的存储。其中各个环节也是冗余的，保证存储的高可用性。

表 7-1 展示了这个环境里面的机器和网络配置：

表 7-1　部分表函数列表

主机名	用途	IP 网卡地址/主机名	ROCE 网卡地址/主机名
PUSCMEB1	MEMBER1	40.43.192.40/PUSCMEB1	5.43.191.40 pm1-en0 5.43.192.40 pm1-en1 5.43.193.40 pm1-en2 5.43.194.40 pm1-en3
PUSCMEB2	MEMBER2	40.43.192.41/PUSCMEB2	5.43.191.41 pm2-en0 5.43.192.41 pm2-en1 5.43.193.41 pm2-en2 5.43.194.41 pm2-en3
PUSCCF01	主 CF	40.43.192.42/PUSCCF01	5.43.191.42 pc1-en0 5.43.192.42 pc1-en1 5.43.193.42 pc1-en2 5.43.194.42 pc1-en3

(续表)

主机名	用途	IP 网卡地址/主机名	ROCE 网卡地址/主机名
PUSCCF02	辅 CF	40.43.192.43/PUSCCF02	5.43.191.43 pc2-en0 5.43.192.43 pc2-en1 5.43.193.43 pc2-en2 5.43.194.43 pc2-en3

表 7-2 展示了这个环境里面的共享磁盘规划：

表 7-2　磁盘规划列表

磁盘系统	磁盘容量	格式	功能
hdisk0	285GB	JFS2 文件系统	操作系统文件卷 rootvg
hdisk2	134GB	GPFS 文件系统	数据库实例共享目录
hdisk3~ hdisk17	1878GB	GPFS 文件系统	数据库的数据目录
hdisk23	134GB	PV	DB2 pureScale 集群仲裁盘

7.2.2　系统环境准备

在安装 DB2 PureScale 前需要对操作系统、以太网络环境、IB 网络配置、共享磁盘阵列配置进行调整和检查，确保 DB2 pureScale 安装条件已经具备。详细的 DB2 pureScale Feature 安装前核对表(AIX)，可参考 IBM 官方文档。

首先需要准备的是检查系统微码级别、操作系统版本是否满足软件需求。在此基础上需要安装 uDAPL 驱动、OpenSSH、C++运行环境等。必须保证环境具备官方文档列举的条件。

7.2.3　配置共享存储

确保所有要用的共享磁盘都已经被操作系统正确识别，并且在所有机器上看到的 PVID 都是一致的，在不同机器上看到的盘号可能不一样。PureScale 至少需要 3 个共享磁盘设备。用 lspv 查看 PVID。

```
#lspv
hdisk0     00f8b37acf4850d6              None
hdisk1     00f8b37acf505406              None
hdisk2     00f8b37acf52f091              None
......
```

```
hdisk21    00f8b37acf476cff                None
hdisk22    00f8b37acf49fc1e                None  )
```

如果不满足，使用如下方法赋予 PVID。在所有节点上分别对 GPFS 磁盘执行如下命令，激活磁盘并生成 PVID：

```
#chdev -l hdisk5 -a pv="yes"
hdiskpower0 changed
```

然后分别对存放实例、数据以及 tie breaker 的磁盘作如下修改：

```
chdev -l hdisk2  -a algorithm=fail over
chdev -l hdisk23  -a algorithm=fail over
chdev -l hdisk3  -a algorithm=round robin
......
chdev -l hdisk17 -a algorithm=round robin

chdev -l hdisk2  -a reserve policy=single path
chdev -l hdisk23 -a reserve policy=single path
chdev -l hdisk3  -a reserve policy=no reserve
......
chdev -l hdisk17 -a reserve_policy=no_reserve
```

7.2.4 配置 IOCP

确认 IOCP 已经安装并且配置好。确认安装的命令如下：

```
PUSCMEB1:/#lslpp -l bos.iocp.rte
  Fileset                      Level  State      Description
------------------------------------------------------------------------
Path: /usr/lib/objrepos
  bos.iocp.rte               7.1.2.15  COMMITTED  I/O Completion Ports API

Path: /etc/objrepos
  bos.iocp.rte               7.1.2.15  COMMITTED  I/O Completion Ports API
PUSCMEB1:/#
```

确认已经配置好的命令如下：

```
# lsdev -Cc iocp
iocp0 Available  I/O Completion Ports
```

注意：如果状态显示为 Defined，可通过 smitty iocp 进行配置。也可以在所有节点上执

行如下命令，配置 IOCP 设备：

```
#mkdev -l iocp0
```

再次验证：

```
# lsdev -Cc iocp
iocp0 Available  I/O Completion Ports
```

7.2.5　配置 RoCE 万兆网络环境

1. 创建 InfiniBand Communication Manager

```
$smitty icm
```

全部选择默认属性，回车确认成功后退出。

```
Infiniband Communication Manager Device Name        icm
Minimum Request Retries                            [1]
Maximum Request Retries                            [7]
Minimum Response Time (msec)                        [100]
Maximum Response Time (msec)                        [4300]
Maximum Number of HCA's                            [256]
Maximum Number of Users                            [65000]
Maximum Number of Work Requests                     [65000]
Maximum Number of Service ID's                      [1000]
Maximum Number of Connections                       [65000]
Maximum Number of Records Per Request                [64]
Maximum Queued Exception Notifications Per User    [1000]
Number of MAD buffers per HCA                       [64]
```

2. 重启主机

```
shutdown -Fr
```

3. 配置/etc/hosts

```
40.43.192.40    PUSCMEB1
40.43.192.41    PUSCMEB2
40.43.192.42    PUSCCF01
40.43.192.43    PUSCCF02

2.43.191.40 pm1-en0
```

```
    2.43.192.40 pm1-en1
    2.43.193.40 pm1-en2
    2.43.194.40 pm1-en3

    2.43.191.41 pm2-en0
    2.43.192.41 pm2-en1
    2.43.193.41 pm2-en2
    2.43.194.41 pm2-en3

    2.43.191.42 pc1-en0
    2.43.192.42 pc1-en1
    2.43.193.42 pc1-en2
    2.43.194.42 pc1-en3

    2.43.191.43 pc2-en0
    2.43.192.43 pc2-en1
    2.43.193.43 pc2-en2
    2.43.194.43 pc2-en3
```

4. 在 PUSCMEB1 上编辑/etc/dat.conf，添加如下内容

```
    hca0 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce0
    1 pm1-en0" " "
    hca1 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr 64.o)
IBM.1.1 "/dev/roce0 2 pm1-en1" " "
    hca2 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce1 1 pm1-en2" " "
    hca3 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce1 2 pm1-en3" " "
```

5. 在 PUSCMEB2 机器上编辑/etc/dat.conf，添加如下内容

```
    hca0 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr 64.o)
IBM.1.1 "/dev/roce0 1 pm2-en0" " "
    hca1 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr 64.o)
IBM.1.1 "/dev/roce0 2 pm2-en1" " "
    hca2 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr 64.o)
IBM.1.1 "/dev/roce1 1 pm2-en2" " "
    hca3 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce1 2 pm2-en3" " "
```

6. 在 PUSCCF01 机器上编辑/etc/dat.conf，添加如下内容

```
    hca0 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce0 1 pc1-en0" " "
    hca1 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce0 2 pc1-en1" " "
    hca2 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce1 1 pc1-en2" " "
    hca3 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr 64.o)
IBM.1.1 "/dev/roce1 2 pc1-en3" " "
```

7. 在 PUSCCF02 机器上编辑/etc/dat.conf，添加如下内容

```
    hca0 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce0 1 pc2-en0" " "
    hca1 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce0 2 pc2-en1" " "
    hca2 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce1 1 pc2-en2" " "
    hca3 u2.0 nonthreadsafe default /usr/lib/libdapl/libdapl2.a(shr_64.o)
IBM.1.1 "/dev/roce1 2 pc2-en3" " "
```

8. 用 lsdev -C | grep -E " Infiniband|PCIE RDMA" 查看 RoCE 万兆网卡状态正确

```
    # lsdev -C | grep -E "Infiniband|PCIe"
    icm       Available          Infiniband Communication Manager
    roce0     Available 02-00      PCIE RDMA over Converged Ethernet RoCE
Adapter
                                (b315506714101604)
```

9. 用 ibstat -v 确认 RoCE 网卡与交换机的连接正常

```
    ------------------------------------------------------------------------
    ETHERNET PORT 1 INFORMATION (roce0)
    ------------------------------------------------------------------------
     Link State: UP
     Link Speed: 10G XFI
     Link MTU: 9600
     Hardware Address: 00:02:c9:4b:97:b8
     GIDS (up to 3 GIDs):
     GID0 :00:00:00:00:00:00:00:00:00:00:00:02:c9:4b:97:b8
```

```
GID1 :00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
GID2 :00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
```

7.2.6 检查文件系统的空间

确保相关文件系统的空余空间：

- 3GB 左右的空间用来解压和存放 DB2 pureScale 安装介质
- 3.5GB 的空间用来安装 DB2 pureScale 软件，安装位置在/db2/software/10.5.2
- 10GB 的/tmp 目录
- 10GB 的 DB2 实例 home 目录/home/db2pure
- 10GB 的/var 目录
- 5GB 用于/(root filesystem)

通过系统命令检查文件系统：

```
PUSCMEB1:/home#df -g
Filesystem    GB blocks     Free %Used    Iused %Iused Mounted on
/dev/hd4           5.00     4.89    3%     7077     1% /
/dev/hd2          12.00     9.06   25%    63080     3% /usr
/dev/hd9var       10.00     9.90    1%      943     1% /var
/dev/hd3          10.00     9.86    2%      410     1% /tmp
/dev/hd1          10.00    10.00    1%       15     1% /home
/dev/hd11admin     0.50     0.50    1%        9     1% /admin
/proc                -        -     -         -      -  /proc
/dev/hd10opt       8.00     7.82    3%     2504     1% /opt
/dev/livedump      0.50     0.50    1%        4     1% /var/adm/ras/livedump
/dev/lvcmbc admin 10.00     9.72    3%       24     1% /cmbc admin
/dev/lvdb2        10.00     4.76   53%     9852     1% /db2
```

7.2.7 配置时钟同步服务

DB2 pureScale 集群中的多个成员节点和 CF 节点一起工作，确保这些节点之间的时间同步是非常重要的。在安装之前，应该配置各成员节点上的 NTP 以保障每个成员节点上的时间一致。客户环境中已经提供了如下时钟源：

```
主时钟源：10.2.95.254
备时钟源：10.2.95.253
```

1. 在客户机的/etc/ntp.conf 文件中添加如下内容

```
driftfile /etc/ntp.drift
```

```
tracefile /etc/ntp.trace
server 40.2.208.64 prefer
server 40.2.208.65
```

2. 编辑/etc/rc.tcpip 文件，将以下条目前面的注释符号#去掉

```
start /usr/sbin/xntpd -x "$src_running"
```

3. 启动 NTP 服务

```
startsrc -s xntpd
```

4. 在客户机上验证 NTP 同步情况

建议先手工修改系统时间，让各个分区的系统时间基本准备好。如果守护程序与系统时钟严重不同步，那么可能需要 10 分钟以上的时间来自动同步时间。

```
#lssrc -ls xntpd
```

5. 为确保在继续之前守护程序已同步，可以通过命令检查“系统层”字段。要继续执行下一个步骤，系统层字段应该小于 16

```
PUSCMEB1:/home#lssrc -ls xntpd
 Program name:    /usr/sbin/xntpd
 Version:        3
 Leap indicator:  00  (No leap second today.)
 Sys peer:       40.2.208.64
 Sys stratum:     3
 Sys precision:   -17
 Debug/Tracing:   DISABLED
 Root distance:   0.001968
 Root dispersion: 0.021072
 Reference ID:    40.2.208.64
 Reference time:  d664f9d5.3c462000  Wed, Dec 25 2013 14:29:41.235
 Broadcast delay: 0.003906 (sec)
 Auth delay:     0.000122 (sec)
 System flags:    bclient pll monitor filegen
 System uptime:   80301 (sec)
 Clock stability: 0.000000 (sec)
 Clock frequency: 0.000000 (sec)
 Peer: 40.2.208.64
```

```
      flags: (configured)(sys peer)(preferred)
      stratum:  2, version: 3
      our mode: client, his mode: server
 Peer: 40.2.208.65
      flags: (configured)(sys peer)
      stratum:  2, version: 3
      our mode: client, his mode: server
Subsystem        Group          PID         Status
 xntpd           tcpip          4260072     active
PUSCMEB1:/home#
```

6. 通过运行 ntpdate -d ntp_server_hostname 命令检查时钟的同步情况

```
PUSCMEB1:/home#ntpdate -d 40.2.208.64
25 Dec 14:35:52 ntpdate[8519908]: 3.4y
transmit(40.2.208.64)
receive(40.2.208.64)
transmit(40.2.208.64)
receive(40.2.208.64)
transmit(40.2.208.64)
receive(40.2.208.64)
transmit(40.2.208.64)
receive(40.2.208.64)
transmit(40.2.208.64)
server 40.2.208.64, port 123
stratum 2, precision -20, leap 00, trust 000
refid [199.0.10.17], delay 0.02606, dispersion 0.00005
transmitted 4, in filter 4
reference time:     d664fb25.576e8294  Wed, Dec 25 2013 14:35:17.341
originate timestamp: d664fb48.531fab96  Wed, Dec 25 2013 14:35:52.324
transmit timestamp:  d664fb48.53199000  Wed, Dec 25 2013 14:35:52.324
filter delay:  0.02696  0.02614  0.02608  0.02606
             0.00000  0.00000  0.00000  0.00000
filter offset: 0.000222 -0.00017 -0.00016 -0.00014
             0.000000 0.000000 0.000000 0.000000
delay 0.02606, dispersion 0.00005
offset -0.000148
```

```
    25 Dec 14:35:52 ntpdate[8519908]: adjust time server 40.2.208.64 offset
-0.000148
    PUSCMEB1:/home#
```

7.2.8　配置用户名和用户组

DB2 pureScale 集群的实例同普通的 DB2 实例一样，需要有实例用户和 Fenced 用户，而且要确保这两个用户在所有的成员和 CF 节点上都存在，并在所有的节点上具有同样的 uid，并且这两个用户所属的 group 在所有节点上的名称和 gid 必须相同。

以本项目为例，确保表 7-3 中的用户和组在所有节点上都存在。

表 7-3　pureScale 用户名和组名配置表

用户类型	用户名	组名
实例用户	db2pure	db2iadm1
Fenced 用户	db2sdfe1	db2fadm1

在各个分区上执行如下步骤以创建所需要的用户组和用户名。创建实例用户和 Fenced 用户的用户组：

```
    mkgroup id=551  db2iadm1
    mkgroup id=102 db2iadm1
```

创建实例用户和 Fenced 用户：

```
    mkuser id=551 pgrp=db2iadm1 groups=db2iadm1 home=/home/db2pure core=-1
data=491519 stack=32767 rss=-1 fsize=-1 db2pure
    echo db2pure:db2pure  | chpasswd -c
```

在其他所有节点上执行上面的命令，确保在所有的节点上创建相同的用户组和用户名并保证 uid 和 gid 一致。分别用新建立的实例用户和 Fenced 用户重新登录系统，如果提示第一次登录要修改密码，可根据要求更改密码。设置实例用户和 Fenced 用户的密码：

```
    passwd db2sdin1
    passwd db2sdfe1
```

7.2.9　配置用户限制

查看当前 ulimit 的参数：filesize 设置为 unlimited，umask 设置为 022。

```
    # id root; ulimit -f; umask
```

```
    uid=0(root) gid=0(system)
groups=2(bin),3(sys),7(security),8(cron),10(audit),11(lp)
    2097151
    022
```

修改 ulimit 设置，编辑/etc/security/limits 文件，修改 root 和实例用户的限制。

```
root:
        core = -1
        data = -1
        stack = -1
        rss = -1
        fsize = -1
db2sdin1:
        core = -1
        data = 491519
        stack = 32767
        rss = -1
        fsize = -1
```

重新登录后检查修改是否成功。

```
# ulimit -f
unlimited
```

7.2.10 配置集群互信

DB2 pureScale 集群中各节点相互之间需用通过 SSH 的方式远程执行命令，因此需要为 root 用户和实例用户配置 SSH 信任关系。

以 root 用户为例，执行以下步骤以配置信任关系。在所有的节点上使用 ssh-keygen 命令以生成公钥/私钥对：

```
ssh-keygen -t rsa -N "" -f $HOME/.ssh/id_rsa
```

该命令在执行过程中会提示用户输入信息，直接按 Enter 键略过。该命令执行完成之后会在~/.ssh 目录下生成密钥文件 id_dsa(私钥)和 id_dsa.pub(公钥)。

将在所有节点上生成的公钥文件的内容合并到一个名为 authorized_keys 的文件中，并将 authorized_keys 拷贝到所有节点的~/.ssh 目录下，使用命令"chmod 644 authorized_keys"修改其权限。登录第一个节点，把所有节点生成的 key 拷贝到第一个节点的 authorized_keys 文件中：

```
cp /.ssh/id_rsa.pub /.ssh/authorized_keys
scp 40.43.192.43:/.ssh/id_rsa.pub /tmp/id_rsa.pub
cat /tmp/id_rsa.pub >> /.ssh/authorized_keys
scp 40.43.192.40:/.ssh/id_rsa.pub /tmp/id_rsa.pub
cat /tmp/id_rsa.pub >> /.ssh/authorized_keys
scp 40.43.192.41:/.ssh/id_rsa.pub /tmp/id_rsa.pub
cat /tmp/id_rsa.pub >> /.ssh/authorized_keys
```

将第一个节点的 authorized_keys 文件分发到其他节点：

```
scp /.ssh/authorized_keys 40.43.192.43:/.ssh/authorized_keys
scp /.ssh/authorized_keys 40.43.192.40:/.ssh/authorized_keys
scp /.ssh/authorized_keys 40.43.192.41:/.ssh/authorized_keys
```

进入每个节点执行，屏蔽掉 SSH 的欢迎信息：

```
touch $HOME/.hushlogin
add /usr/lpp/mmfs/bin to $PATH
```

互相测试：

```
ssh 40.43.192.42 date
ssh 40.43.192.43 date
ssh 40.43.192.40 date
ssh 40.43.192.41 date
```

注意，一定要保证用户的 home 目录和~/.ssh 目录不开放 group 和 others 的写权限，否则 SSH 认为这是一个安全漏洞，信任关系会被拒绝。

该配置需要为 root 和 instance 用户都执行。root 和 instance 用户的 home 目录都已生成 authorized_keys 合成公钥文件，需要将每个节点的主机名添加到/etc/hosts 中，再次执行以下命令创建节点之间的互信。

可以在所有节点上手动执行 ssh 命令来验证 SSH 信任关系是否配置成功：

```
ssh 40.43.192.42 date
ssh 40.43.192.43 date
ssh 40.43.192.40 date
ssh 40.43.192.41 date
```

7.2.11　执行安装检查

在所有的节点机器上运行 db2prereqcheck 命令以保证其符合 DB2 pureScale 安装的先决

条件。检查基本通过，除了/home 空间检查不够。但是，如果实例用户目录不在此目录下，可以忽略。

```
    PUSCMEB1:/InstMedia/DB2/10.5/server#./db2prereqcheck
    ==================================================================
    Checking DB2 prerequisites for DB2 database version "10.5.0.2" on operating
system "AIX"

    Validating "kernel level " ...
      Required minimum operating system kernel level: "6.1".
      Actual operating system kernel level: "7.1".
      Requirement matched.

    ......

    Validating "uDAPL" ...
      Required minimum "uDAPL" level:  "7.1.1.15"
      Actual version:  "7.1.1.15"
      Requirement matched.
    DBT3533I  The db2prereqcheck utility has confirmed that all installation
prerequisites were met for DB2 database "server " "with DB2 pureScale feature
".  Version: "10.1.0.2".
```

7.2.12 安装 DB2 pureScale 软件

使用 db2_install 命令安装 DB2 软件到路径/db2/software/10.5.2 下：

```
     # ./db2_install
    DBI1324W  Support of the db2_install command is deprecated. For
         more information, see the DB2 Information Center.


    Default directory for installation of products - /opt/IBM/db2/V10.5
    ***********************************************************
    Install into default directory (/opt/IBM/db2/V10.5) ? [yes/no]
    no
    Enter the full path of the base installation directory:
    ------------------------------------------------
    /db2/software/10.5.2
```

```
    Specify one of the following keywords to install DB2 products.
      AESE
      ESE
      CONSV
      WSE
      CLIENT
      RTCL

    Enter "help" to redisplay product names.
    Enter "quit" to exit.
    ***********************************************************
    AESE
    ***********************************************************
    Do you want to install the DB2 pureScale Feature? [yes/no]
    yes
    Requirement not matched for DB2 database "Server" "with pureScale feature".
Version: "10.1.0.2".
    Summary of prerequisites that are not met on the current system:
    DBT3571E  The db2prereqcheck utility found that the directory "/home" does
not have enough free space on host "p720cp2m". Required space: "5120000 KB".
Actual space present: "99444 KB".


    DB2 installation is being initialized.

     Total number of tasks to be performed: 49
    Total estimated time for all tasks to be performed: 2691 second(s)

    Task #1 start

    ......

    The execution completed successfully.

    For more information see the DB2 installation log at
    "/tmp/db2_install.log.11599970".
```

7.2.13　安装 DB2 许可

使用 db2licm 命令安装和检查 DB2 软件许可。

安装 pureScale 许可:

```
    PUSCMEB1:/#/db2/software/10.5.2/adm/db2licm -a
/InstMedia/DB2/License_10.5/DB2_10.5_license/db2aese_c.lic
    LIC1402I  License added successfully.
    LIC1426I  This product is now licensed for use as outlined in your License
Agreement.  USE OF THE PRODUCT CONSTITUTES ACCEPTANCE OF THE TERMS OF THE IBM
LICENSE AGREEMENT, LOCATED IN THE FOLLOWING DIRECTORY:
"/db2/software/10.5.2/license/en_US.iso88591"
    You have mail in /usr/spool/mail/root
    PUSCMEB1:/#/db2/software/10.5.2/adm/db2licm -a
/InstMedia/DB2/License_10.5/DB2_10.5_license/db2aese_c.lic
    PUSCMEB1:/InstMedia/DB2/License_10.5/DB2_10.5_license#/db2/software/10.
5.2/adm/db2licm -l
    Product name:                  "DB2 Advanced Enterprise Server Edition"
    License type:                  "CPU Option"
    Expiry date:                   "Permanent"
    Product identifier:             "db2aese"
    Version information:            "10.5"
    Enforcement policy:             "Soft Stop"
```

7.2.14 创建实例

使用 db2icrt 命令创建包含一个成员和一个 CF 的实例:

```
    PUSCMEB1:/#/db2/software/10.5.2/instance/db2icrt  -d -p 50000 -u db2pure -m
PUSCMEB1 -mnet pm1-en0,pm1-en1,pm1-en2,pm1-en3 -cf PUSCCF01 -cfnet
pc1-en0,pc1-en1,pc1-en2,pc1-en3 -instance_shared_dev /dev/hdisk2
-instance_shared_mount /db2home_shared -tbdev /dev/hdisk23 db2pure
    DBI1446I  The db2icrt command is running.

    DB2 installation is being initialized.

     Total number of tasks to be performed: 10
    Total estimated time for all tasks to be performed: 1074 second(s)

    ......

    The execution completed successfully.

    For more information see the DB2 installation log at
```

```
    "/tmp/db2icrt.log.5701848".
    Required: Review the following log file also for warnings or errors:
    "/tmp/db2icrt local.log.*"
    DBI1070I  Program db2icrt completed successfully.
```

使用 db2iupdt 命令加入一个 CF：

```
    PUSCMEB1:/#/db2/software/10.5.2/instance/db2iupdt -add -cf PUSCCF02 -cfnet
pc2-en0,pc2-en1,pc2-en2,pc2-en3 db2pure

    DBI1446I  The db2iupdt command is running.


    DB2 installation is being initialized.

     Total number of tasks to be performed: 10
    Total estimated time for all tasks to be performed: 1074 second(s)

    ......

    The execution completed successfully.

    For more information see the DB2 installation log at
    "/tmp/db2iupdt.log.5832982".
    DBI1070I  Program db2iupdt completed successfully.
```

7.2.15　配置实例

切换到实例用户，修改 db2 注册表变量，配置通信协议和字符集：

```
    db2set DB2_ALLOW_WLB_WITH_SEQUENCES=yes
    db2set DB2 OPTSTATS LOG=ON,NUM=10,SIZE=100
    db2set DB2FODC=DUMPCORE=OFF
    db2set DB2_SKIPINSERTED=ON
    db2set DB2 EVALUNCOMMITTED=ON
    db2set DB2_EXTENDED_OPTIMIZATION=ON
    db2set DB2 ANTIJOIN=ON
    db2set DB2COMM=TCPIP
    db2 update dbm cfg using SVCENAME db2c db2pure
    db2 update dbm cfg using CF MEM SZ 9751756
    /db2/software/10.5.2/bin/db2cluster -cm -set -tiebreaker -disk
```

```
/dev/hdisk23
```

修改 db2 实例参数(注意端口 60000 供 FCM 内部通信使用，不要使用这个端口作为服务端口)。

切换到实例用户，使用 db2start 命令启动实例：

```
    PUSCMEB1:/home/db2pure$db2start
    01/06/2014 13:51:50     0   0   SQL1063N  DB2START processing was
successful.
    01/06/2014 13:51:52     1   0   SQL1063N  DB2START processing was
successful.
    SQL1063N  DB2START processing was successful.
```

使用 db2instance 命令查看实例状态：

```
    PUSCMEB1:/home/db2pure$db2instance -list
    ID       TYPE            STATE              HOME_HOST
CURRENT HOST           ALERT   PARTITION NUMBER     LOGICAL PORT    NETNAME
    --      ----       -----    ----------         ------------     ---------
    0      MEMBER          STARTED              PUSCMEB1
PUSCMEB1             NO                0                0
pm1-en0,pm1-en1,pm1-en2,pm1-en3
    1      MEMBER          STARTED              PUSCMEB2
PUSCMEB2             NO                0                0
pm2-en0,pm2-en1,pm2-en2,pm2-en3
    128     CF             PRIMARY              PUSCCF01
PUSCCF01             NO                -                0
pc1-en0,pc1-en1,pc1-en2,pc1-en3
    129     CF             CATCHUP              PUSCCF02
PUSCCF02             NO                -                0
pc2-en0,pc2-en1,pc2-en2,pc2-en3

    HOSTNAME                STATE             INSTANCE STOPPED      ALERT
    --------                -----             ----------------      -----
    PUSCCF02               ACTIVE                        NO          NO
    PUSCCF01               ACTIVE                        NO          NO
    PUSCMEB2               ACTIVE                        NO          NO
    PUSCMEB1               ACTIVE                        NO          NO
```

7.2.16　创建 GPFS 文件系统

在安装目录中，使用 db2cluster 命令创建 db2data 文件系统：

```
PUSCMEB1:/#/db2/software/10.5.2/bin/db2cluster -cfs -create -filesystem
db2data -disk /dev/hdiskpower4,/dev/hdiskpower5,/dev/hdiskpower6,/dev/hdiskpower7,
/dev/hdiskpower8,/dev/hdiskpower9,/dev/hdiskpower10,/dev/hdiskpower11,/dev/
hdiskpower12,/dev/hdiskpower13,/dev/hdiskpower14,/dev/hdiskpower15 -mount
/db2data File system 'db2data' has been successfully created.
```

切换至 root 用户，为上述 GPFS 文件系统赋权：

```
PUSCMEB1:/#chown -R db2pure:db2iadm1 /db2data
```

7.2.17　创建数据库

切换到实例用户，创建数据库 SAMPLE：

```
db2 create db chgmdb on /db2data/chgmdb/data dbpath on /db2data/chgmdb using
codeset utf-8 territory cn  pagesize 16 K
```

7.3　DB2 集群的维护

DB2 pureScale 集群与传统单节点 DB2 的管理方式略有不同。本节主要介绍 pureScale 集群相关的常用运维命令。

7.3.1　实例的启停

1) 查看实例的状态

获取 member 和 CF 的当前状态是非常重要的，所有成员都必须处于“started”状态，而其中一个 CF 应该处于“PRIMARY”状态，其余 CF 处于“PEER”状态。这里不应该有任何警告；换句话说，在 ALERT 列都应该是“NO”。集群的所有主机都必须是“ACTIVE”。

```
# db2instance -list
```

2) 启动实例

使用命令行启动实例，输入：

```
db2start
```

3) 停止实例

使用命令行来停止实例，请输入：

```
db2stop
```

4) 仅启动指定成员

使用命令行来启动指定成员，请输入：

```
db2start member 1
```

5) 仅停止指定成员

使用命令行来停止指定成员，请输入：

```
db2stop member 1
```

6) 仅启动指定 CF

使用命令行来启动指定 CF，请输入：

```
db2start cf 129
```

7) 仅停止指定 CF

使用命令行来停止指定 CF，请输入：

```
db2stop cf 129
```

8) 强制停止

在停止成员或整个集群之前，需要先停止连接至该节点的应用。如果需要强制停止，可以加 force 选项。例如：

```
db2stop member 0 force
```

9) 查看应用连接

请输入：

```
db2_all "db2 list applications"
```

7.3.2 集群的管理

DB2 集群包含 TSA 集群和 GPFS 文件系统集群。这两个集群软件有很多的工具用于管理，功能也很丰富。对于维护数据库的管理员，DB2 提供了管理工具 db2cluster 来帮助管理 DB2 pureScale 集群。

1) 查看所有实例的状态、角色、所处的主机名、RDMA 通信端口

```
#db2instance -list
```

2) 显示当前 TSA 的仲裁盘

```
#db2cluster -cm -list -tiebreaker
```

3) 设置 TSA 的仲裁盘

```
#db2cluster -cm -set -tiebreaker -disk <diskname>
```

4) 设置当前为集群管理节点

```
#db2cluster -cm -set -tiebreaker -majority
```

5) 查看检测主机的心跳时间

```
#db2cluster -cm -list -hostfailuredetectiontime
```

6) 设置成员节点主机的心跳检测时间

```
#db2cluster -cm -set -option -hostfailuredetectiontime -value <value>
```

7) 显示初始化的主 CF

```
#db2cluster -cm -list -pprimary
```

8) 设置哪个 CF 为初始化的主 CF

```
#db2cluster -cm -set -option -pprimary -value <value>
```

9) 开启或关闭 member 的自动回飘

```
#db2cluster -cm -set -option -autofailback -value <on|off>
```

10) 查看实例的警告

```
# db2cluster -cm -list -alert
```

11) 显示当前系统的 TSA 版本

```
# db2cluster -cm -list -localhostversion
```

12) 查看 member 是否开启了自动回切

```
#db2cluster -cm -list -autofailback
```

13) 判断当前资源模型是否正常，是否保持一致

```
#db2cluster -cm -verify -resources
```

14) 判断当前集群管理器是否处于维护中

```
#db2cluster -cm -verify -maintenance
```

15) 让当前节点进入维护中

```
#db2cluster -cm -enter -maintenance
```

16) 让所有节点进入维护中

```
#db2cluster -cm -enter -maintenance -all
```

17) 让当前节点退出维护状态

```
#db2cluster -cm -exit -maintenance
```

18) 提交对集群的修改

```
#db2cluster -cm -commit
```

19) 清除警告

```
#db2cluster -cm -clear -alert
```

20) 显示集群管理器的域

```
#db2cluster -cm -list -domain
```

21) 创建集群管理器的域

```
#db2cluster -cm -create -domain <name> -host <hostname>
```

22) 在集群管理器的域中加入新成员

```
#db2cluster -cm -add -host <hostname>
```

23) 在集群管理器的域中删除成员

```
#db2cluster -cm -remove -host <hostname>
```

24) 显示集群管理器的域中成员的状态

```
#db2cluster -cm -list -host -state
```

25) 删除集群管理器的域

```
#db2cluster -cm -delete -domain <domainname>
```

26) 修复集群管理器的域

```
#db2cluster -cm -repair -domain <domainname>
```

27) 启停集群管理器的域

```
#db2cluster -cm <-start|-stop> -domain <domainname>
```

28) 启停集群管理器的域中的成员

```
#db2cluster -cm <-start|-stop> -host <hostname>
```

29) 创建集群管理器的资源模型

```
#db2cluster -cm -create -resources
```

30) 删除集群管理器的资源模型

```
#db2cluster -cm -delete -resources
```

31) 修复集群管理器的资源模型

```
#db2cluster -cm -repair -resources
```

32) 创建 GPFS，多块磁盘用逗号分隔

```
#db2cluster -cfs -create -filesystem <fsname> -disk <disklist> -mount <mount
directory>
```

33) 给当前 GPFS 加盘

```
#db2cluster -cfs -add -filesystem <fsname> -disk <disklist>
```

34) 从当前 GPFS 中卸盘

```
#db2cluster -cfs -remove -filesystem <fsname> -disk <diskname>
```

35) 删除 GPFS

```
#db2cluster -cfs -delete -filesystem <fsname>
```

36) 设置 GPFS 的仲裁盘

```
#db2cluster -cfs -set -tiebreaker -disk <diskname>
```

37) 设置当前节点为 GPFS 管理节点

```
#db2cluster -cfs -set -tibreaker -majority
```

38) 显示当前 GPFS 的仲裁盘

```
#db2cluster -cfs -list -tiebreaker
```

39) 显示当前 GPFS 的物理盘

```
#db2cluster -cfs -list -filesystem <fsname> -disk
```

40) 显示 GPFS 的配置

```
#db2cluster -cfs -list -filesystem <fsname> -configuration
```

41) 显示当前 GPFS 版本

```
#db2cluster -cfs -list -localhostversion
```

42) 验证 GPFS 的配置是否正确

```
# db2cluster -cfs -verify -configuration
```

43) 验证 GPFS 是否处于维护中

```
# db2cluster -cfs -verify -maintenance
```

44) 挂载 GPFS

```
# db2cluster -cfs -mount -filesystem <fsname>
```

45) 卸载 GPFS

```
# db2cluster -cfs -umount -filesysytem <fsname>
```

46) 重新平衡 GPFS 下的数据在磁盘中的分布

```
#db2cluster -cfs -rebalance -filesystem <fsname>
```

47) 让当前 GPFS 节点进入维护中

```
#db2cluster -cfs -enter -maintenance
```

48) 让所有 GPFS 节点进入维护中

```
#db2cluster -cfs -enter -maintenacen -all
```

49) 启动整个 GPFS

```
#db2cluster -cfs -start -all
```

50) 启动个别 GPFS 节点

```
#db2cluster -cfs -start -host <hostname1,hostname2…>
```

51) 停止整个 GPFS

```
#db2cluster -cfs -stop -all
```

52) 停止个别 GPFS 节点

```
#db2cluster -cfs -stop -host <hostname1,hostname2…>
```

7.3.3　故障处理

当 DB2 集群出现问题的时候，需要结合从操作系统、DB2、TSA 和 GPFS 四个方面收集的信息来诊断。

1. DB2 pureScale 数据库工具

DB2 提供的 db2instance 工具用于查看 DB2 pureScale 集群的状态。使用实例用户执行“db2instance -list”：

```
    AGDPCMB1:/home/db2gdpc$db2instance -list
    ID        TYPE              STATE                 HOME HOST
CURRENT HOST           ALERT   PARTITION NUMBER       LOGICAL PORT    NETNAME
    --        ----              -----                 ---------
------------           -----   ----------------       ------------    -------
    0      MEMBER            STOPPED                 AGDPCMB1
AGDPCMB1              NO                   0                   0    am1-0,am1-1
    1      MEMBER            STOPPED                 AGDPCMB2
AGDPCMB2              NO                   0                   0    am2-0,am2-1
```

```
    2       MEMBER          STOPPED                  BGDPCMB1
BGDPCMB1             NO                0                  0    bm1-0,bm1-1
    3       MEMBER          STOPPED                  BGDPCMB2
BGDPCMB2             NO                0                  0    bm2-0,bm2-1
    128     CF              STOPPED                  AGDPCCF1
AGDPCCF1             NO                -                  0    ac1-0,ac1-1
    129     CF              STOPPED                  BGDPCCF1
BGDPCCF1             NO                -                  0    bc1-0,bc1-1

    HOSTNAME                STATE               INSTANCE STOPPED      ALERT
    --------                -----               ----------------      -----
    BGDPCCF1               ACTIVE                            NO          NO
    AGDPCCF1               ACTIVE                            NO          NO
    BGDPCMB2               ACTIVE                            NO          NO
    BGDPCMB1               ACTIVE                            NO          NO
    AGDPCMB2               ACTIVE                            NO          NO
    AGDPCMB1               ACTIVE                            NO          NO
```

如上所示，STATE 列显示了所有 member、CF 和主机的状态。用户很容易在这里发现集群环境是否有问题。ALERT 列会提示用户集群环境是否存在警告。如果有警告，这个工具会提示使用 db2cluster 工具来查看和清理。

2. DB2 pureScale 数据库日志

数据库的日志文件 db2diag.log 会记录集群在运行过程中出现的问题。这个文件的位置由 DIAGPATH 实例参数设置。DIAGLEVEL 实例参数控制收集诊断信息的级别。CF 的诊断信息 cfdiag*.log 会放在 CF_DIAGPATH 设置的路径里。默认 CF_DIAGPATH 和 DIAGPATH 一致。

3. TSA 查看集群资源工具

因为集群软件 TSA 监测 DB2 pureScale 集群的所有资源，所以获取所有资源状态的最好工具还是集群软件 TSA 提供的 lssam 工具。

```
    AGDPCMB1:/home/db2gdpc$lssam
    Pending online IBM.ResourceGroup:ca_db2gdpc_0-rg Binding=Sacrificed
Nominal=Online
           '- Offline IBM.Application:ca_db2gdpc_0-rs Control=StartInhibited
           |- Offline IBM.Application:ca_db2gdpc_0-rs:AGDPCCF1
           '- Offline IBM.Application:ca_db2gdpc_0-rs:BGDPCCF1
```

```
......
        |- Online IBM.PeerNode:AGDPCCF1:AGDPCCF1
        '- Online IBM.PeerNode:BGDPCCF1:BGDPCCF1
```

lssam 列举了所有集群的资源状态，甚至还可以展示各个资源之间的关系，例如共存还是互斥等。通过这个工具可以快速定位问题所在的位置。

4. TSA 诊断日志

TSA 关于 DB2 pureScale 的诊断日志放在/var/ct/<domain_name>/log/mc/IBM.GblResRM/下，可以用于进一步诊断产生异常状态的起因。

5. GPFS 工具和日志

GPFS 是成熟的共享文件系统产品，提供了很多工具用以定位共享存储这块的问题，例如 mmlsnode。GPFS 工具一般放在/usr/lpp/mmfs/bin 下，日志可在/var/mmfs 下找到。

6. uDAPL 工具(Infiniband)

IB 网络作为特殊的硬件环境，可以使用自己的 uDAPL 工具来查看问题。例如使用 ibstat 查看 IB 网卡和网络的状态。

7. 系统工具

分析 DB2 pureScale 数据库的问题还需要结合操作系统提供的工具和日志。例如使用 ifconfig 可以查看网络，使用 iostat 可以查看磁盘读取等。在 AIX 操作系统里可以使用 errpt 查看操作系统的异常信息。

因为 DB2 pureScale 集群环境用到如此多的资源，所以需要将 DB2 和其他这些工具结合起来分析和解决问题。

7.4　DB2 集群设计调优

DB2 集群与单机版相比最大的开销就是不同成员之间的事务存在竞争。也就是说，热点数据的处理问题在 DB2 集群环境中被放得很大。基本上 DB2 集群的调优手段都是围绕着如何打散热点数据页来进行的。

7.4.1 使用小的 pagesize

DB2 集群推荐使用 pagesize 小的表空间。尤其是热点表，如果查询、更新、插入操作比较多，就会在成员之间产生竞争。小一点的 pagesize 使一个页面放入的数据少，产生竞争的概率变小。

7.4.2 使用大的 extentsize

表空间使用了小的 pagesize，为了提高预取的效率，最好使用大的 extentsize。尤其是在有大量插入的场景中，能够观察到使用大的 extentsize 效果更好。

7.4.3 使用 lob inline 方法

DB2 数据库提供了 lob inline 方式来存放大对象。DB2 默认大对象是和行数据分开存放的。大对象的读取和写入也不走缓冲池，在数据库内部记录为 DirectRead 和 DirectWrite。但是有很多大对象数据其实并不是很大，完全可以放在一个页面内，和行数据放在一起。这样读写性能高，同时还能被压缩。lob inline 方式就是为表定义一个 INLINE LENGTH，所有小于这个值的大对象都和数据存放在一起。

对于数据库集群，lob inline 方式意味着更大的行记录，单个页面存放更少的行，减少了竞争概率。

7.4.4 使用大的 pctfree 设置

DB2 的表和索引都可以设置 pctfree，这个值是百分比，意思是在一个页面内部预留多少空间。这个参数在 load 和 reorg 的时候才生效，对正常的插入是没有作用的。所以通常情况下并不建议修改。同时设置更大的 pctfree，意味着需要更多的页来装数，也就是空间占用高。所以这个方法的适用场景要限制好：

- 小表：小表的空间占用少，即便设置大的 pctfree，也不会导致空间紧张。
- 热表：数据查询更新很多，并且集中，需要分散热点页。
- 插入量小：插入数据会塞满 pctfree 的空间，所以插入量大的表并不适合。

7.4.5 巧用 CURRENT MEMBER

DB2 在多节点环境中提供了 CURRENT MEMBER 变量，若应用连接在成员 0 节点，那么获取的这个 CURRENT MEMBER 变量就是 0。

```
AGDPCMB1:/chgm_db2arc/kzh$db2 "values current member"

1
-----------
          0

1 record(s) selected.
```

如果为表加入一列并插入 CURRENT MEMBER 的值，那么不同成员插入的值就不一样。首先，CURRENT MEMBER 的用处是解决热点索引页。

这里讨论两种不同类型的索引。一种索引是插入频繁的、顺序增长的数字，例如自增列、序列号、时间戳等。因为追加的行都落在索引的最后几页，会导致热点页问题。通过加入 CURRENT MEMBER 到索引里，每个成员插入的索引会在不同的页里。示例如下：

```
    alter table orders add column curmem smallint default current member
implicitly hidden;
    create index seqindex on ordernumber (curmem, seqnumber);
```

orders 表有个顺序增长的 seqnumber 列，加入 curmen 之后，不同成员插入的 curmem 不同，对应的索引页也不同，因此解决了热点索引页问题。

另一种索引类型是索引取值比较少，每次插入新的数据都是在原有的索引值上插入新的 RID，导致这些值对应的索引页称为热点。例如省份、邮编这样的列。这种情况下，您可以把 CURRENT MEMBER 加到索引的后面，也可以促使不同成员插入的数据落在不同的索引页面里。示例如下：

```
    alter table customer add column curmem smallint default current member
implicitly hidden;
    create index stateidx on customer (state, curmem);
```

7.4.6　巧用随机索引

随机索引是 DB2 为了解决热点页问题提供的新功能。DB2 的索引一般都是升序或降序排列。DB2 从 V10 开始提供了随机索引。简单来说，随机索引就是将索引的列值按照哈希算法分裂，然后对同样的哈希值页面再排序。所以随机索引对于按照值定向查询是很快的，但是对于范围取值的效率会低于常规索引。

示例如下：

```
CREATE TABLE "DB2PURE"."SMS_CODE"  (
              "SC_TASKID" VARCHAR(32 OCTETS) NOT NULL ,
              "SC_MOBILE" VARCHAR(20 OCTETS) ,
              "SC CODE" VARCHAR(40 OCTETS) ,
              "SC_COUNT" INTEGER ,
              "SC_CREATETIME" TIMESTAMP ,
              "SC_EXPIREDTIME" TIMESTAMP)
             ORGANIZE BY ROW;

CREATE UNIQUE INDEX "DB2PURE"." SC_CREATETIME" ON "DB2PURE"."SMS _CODE"
             ("SC_CREATETIME" RANDOM)
             INCLUDE NULL KEYS ALLOW REVERSE SCANS;
```

上面介绍了在 DB2 集群里一些常用的调优方法，中心思想就是通过数据库内部的技术手段打散热点数据页，减少成员之间的竞争。事实上从上层应用角度考虑，通过区分不同的工作负载，指定有竞争关系的负载运行在相同的成员节点上，也可以避免成员之间的竞争。当然这种方式与应用设计的关系比较密切，需要因地制宜。

7.5　同城双活集群介绍

DB2 集群采用的是共享存储的方式，所以通常 DB2 pureScale 集群都是架建在同一地理位置。但是这种情况下，DB2 pureScale 集群部署在同一站点，如果这个站点整体发生故障，数据库集群将不可用。

针对这种站点级别高可用的需求，DB2 提供了一种特殊的地理上分散的 DB2 pureScale 集群架构解决方案，也就是 GDPC。所以 GDPC 只是搭建 DB2 pureScale 集群的一种方式，逻辑上还是一套 pureScale 集群，但是物理上分散在不同地理位置。这样即使其中一个站点整体故障，DB2 pureScale 集群也只会损失一半的节点，还有一半的节点存活，仍然可以提供数据库服务。

下面通过图 7-4 来了解 GDPC 同城双活集群的架构：

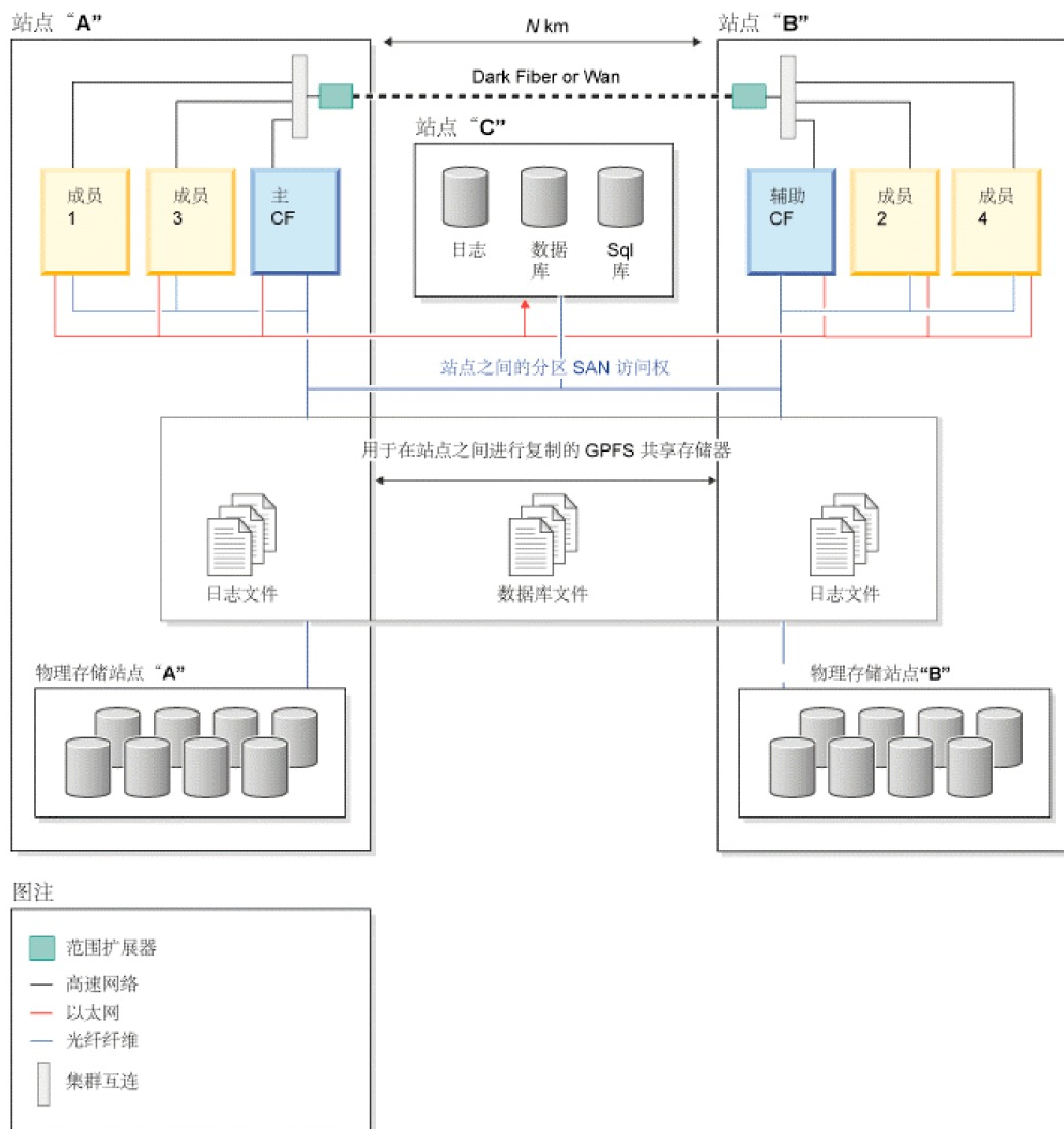


图 7-4　同城双活集群的架构

如上图所示，典型的 GDPC 架构是三站点的方式。站点 A 和站点 B 是对等站点，真正提供数据库服务。站点 C 是仲裁站点，当站点 A 和站点 B 之间网络出现问题的时候，也就是有可能出现所谓的脑裂的情况下，谁抢到了站点 C，谁就是存活的站点。这样可以避免脑裂的发生。

站点 A 和站点 B 都需要具备独立存活的能力。所有集群里的两个 CF 会分布在这两个站点上，集群里的所有 member 必须是双数，同时必须一半一半地分布在这两个站点上。

每个站点都有自己的本地存储，两个站点间存储需要进行实时复制。这样即便一个站点出现问题，另外一个站点的存储还是能够一样使用。所以这两个站点几乎可以称为是完全对称的。

站点 C 作为仲裁站点，平时是不工作的，所以没有特殊的性能需求。它只会在出现需要做仲裁的时候起作用。站点 C 也需要设计本地存储，加入到 GPFS 集群里面，作为仲裁盘，不需要放置任何数据，所以对性能也没有要求。

站点 A 和站点 B 之间需要使用 RDMA 协议通信，还需要存储复制。这些功能都是由 DB2 pureScale 来实现，但是也需要硬件的支持。所以 A、B 站点之间的网络、存储的 SAN 网络需要相通。这是相对于传统 pureScale 本地集群，实现 GDPC 代价比较高的地方。A、B 站点的距离和网络带宽，会直接影响到数据库的性能。这也是 GDPC 环境最需要关注的地方。

7.6 DB2 集群异地容灾

DB2 pureScale 特性实现了同机房的高可用解决方案。GDPC 实现了同城双中心数据库双活的高可用容灾方案。在单机版的 DB2 实例环境中，HADR 是异地容灾的最佳解决方案。从 DB2 V10.5 开始，DB2 pureScale 特性与 HADR 特性可以结合在一起使用，满足两地三中心容灾需求。DB2 pureScale 是典型的数据库集群架构，多个成员同时为一个数据库服务。每个成员会记载各自的日志。HADR 是基于日志传输的同步手段，在存在多个成员日志的情况下，备集群的成员需要合并日志流，然后重放，从而达到同步的功能。

7.6.1 DB2 集群异地容灾架构

在 DB2 pureScale 集群+HADR 容灾架构的体系中，备端也需要是 DB2 pureScale 集群，并且备集群的成员数量要大于或等于主集群的成员数量。这是因为在现有的体系架构下，对于主集群的所有成员 ID，备集群需要有同样 ID 的成员结构才能做日志回放。

7.6.2 Replay Member 概念

从图 7-5 中可以看出 DB2 pureScale 环境下 HADR 的实现机制。在备集群，只有一个成员会执行重做工作。所有的日志流都通过 TCP/IP 网络传输至此成员。所以选取合适的 Replay Member 非常重要。备集群的 Replay Member 可以安排资源比较充分的节点。例如可以选择分配更多的 CPU 和内存资源给这个 LPAR。

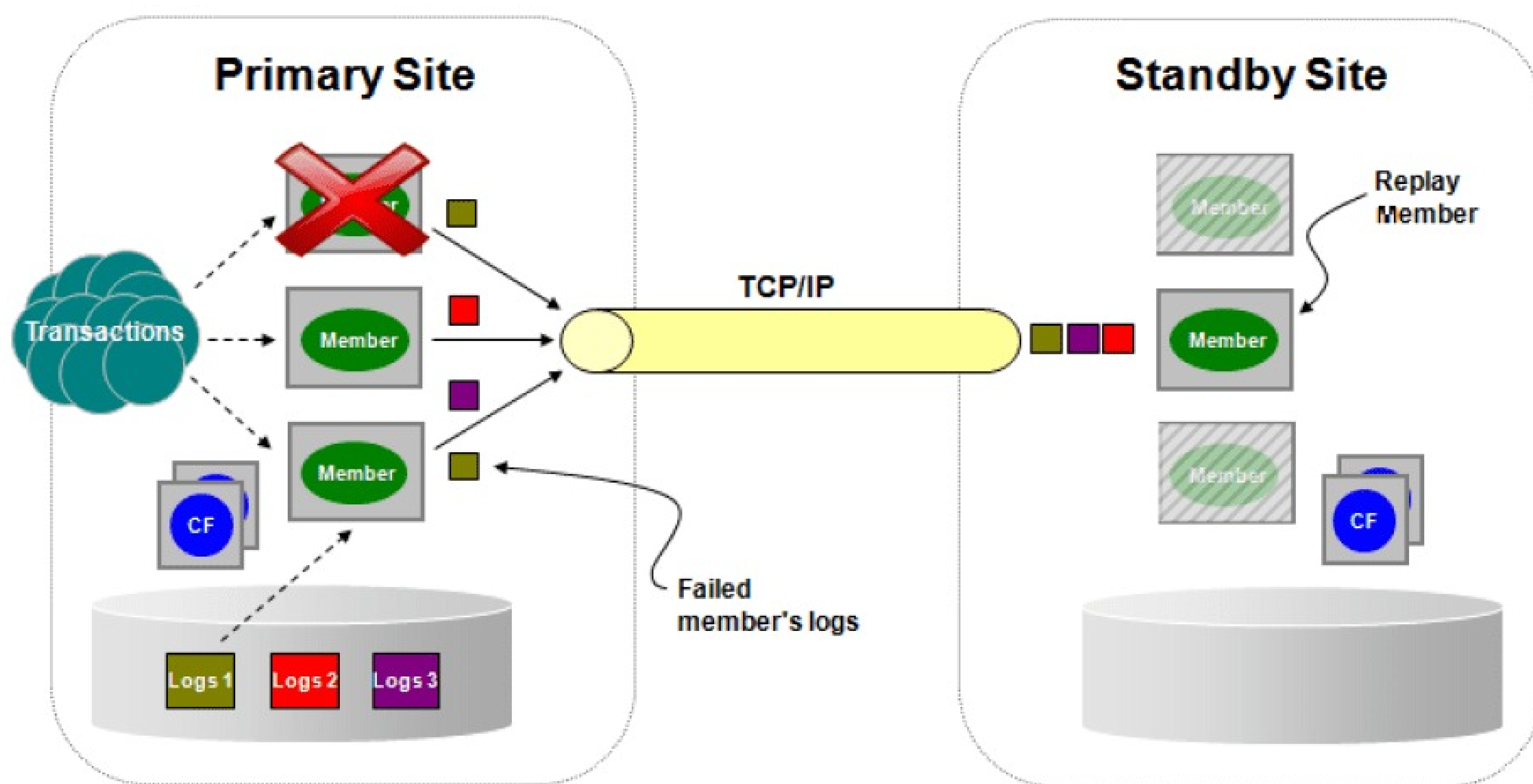


图 7-5　pureScale + HADR 逻辑架构

Replay Member 是一个逻辑上的概念，仅仅是指在做回放的成员。用户可以指定偏好的成员成为 Replay Member。又称之为 Prefer Member，意思是尽可能选择在这个偏好的成员上做 Replay Member。

在 DB2 pureScale 集群+HADR 的容灾架构中，备集群的 Replay Member 上数据库是激活并工作的。在其他的成员上该数据库处于非激活状态。Replay Member 这个角色在 pureScale 集群里面同样具有高可用性。一旦当前的 Replay Member 出现故障，DB2 会在集群里面的其他成员上选择一个激活数据库，成为新的 Replay Member。在选择的过程中，如果有偏好的成员，就会选择偏好的成员。如果发生故障的是偏好的成员，则随机选择一个。所以偏好的成员并不是强制的概念。Replay Member 也可以是任何一个成员。

7.6.3　DB2 集群异地容灾同步模式

HADR 同步方式确定高可用性灾难恢复(HADR)数据库解决方案对事务损失的保护程度。同步方式决定主数据库服务器根据备用数据库上记录的状态，认为何时事务已完成。以同步方式配置参数值越严格，数据库解决方案就越能防止事务数据丢失，但事务处理速度也越慢，所以必须在防止事务丢失的需求与性能需求之间取得平衡。

HADR 同步模式主要有四种：SYNC(同步)、NEARSYNC(接近同步)、SUPERASYNC(超级异步)和 ASYNC(异步)。

DB2 pureScale 集群解决本地高可用，异地容灾通过 HADR 完成。异地 HADR 一般选择异步和超级异步两种模式，需要考虑数据丢失的可能性。因为主数据库上的事务落实操作不会受到相对较慢的 HADR 网络或备用 HADR 服务器的影响，所以主数据库与备用数

据库之间的日志间隔可能会继续增大。监视日志间隔很重要，这是因为在主系统上发生真正的灾难时，日志间隔是对可能丢失的事务数的间接度量。在灾难恢复方案中，日志间隔期间落实的任何事务都对备用数据库不可用。因此，请使用 hadr_log_gap 监视元素来监视日志间隔；如果出现日志间隔不可接受的情况，请调查网络中断次数或备用数据库节点的相对速度，并采取纠正措施以减小日志间隔。

7.6.4 DB2 集群异地容灾切换方式

与普通单机版 DB2 数据库的 HADR 功能一样，DB2 pureScale 集群下的 HADR 功能延续了简单易用的切换方式。仅仅是一条 takeover 命令，就可以完成整个数据库的主备切换。切换的效率非常高，同时切换过程中也进行数据的保护，杜绝各种脑裂情况的发生。DB2 pureScale 集群下的 HADR 切换也分为正常切换和强制切换两种。

在主备集群相连通的情况下，正常切换会首先关闭原主端的数据库，备端接受全部日志并回放完成后变为新的主端，原主端会激活数据库变为新的备端，整个切换过程就完成了，数据库也可以提供服务了，HADR 也正常工作。

在主备集群相连通的情况下，强制切换会强行关闭原主端的数据库并阻止它激活，并且没有改变它的 HADR 角色属性。然后备端接受全部日志并回放完成后变为新的主端，整个切换过程就完成了，数据库也可以提供服务了，HADR 处于非启动状态。

在主备集群不连通的情况下，备端感受不到主端的状态。这种情况只能进行强制切换。强制切换会将备端在日志回放完成后变为新的主端，数据库提供服务，HADR 处于非启动状态。一旦与原主端的网络恢复，就会出现脑裂的场景。DB2 会选择杀掉原主端，从而保证不出现脑裂的现象。

7.6.5 DB2 集群异地容灾客户端连接方式

DB2 pureScale 集群的实现机制里面使用的是 TSA 集群软件，基础是 RSCT。一台机器只能有一个 RSCT 集群，所以采用先前通用的 HACMP 等虚拟 IP 方式来配置应用以连数据库已经不可行了。如果是虚拟 IP，对于 pureScale 集群来说也很难做到负载均衡等各种连接方式。所以在 DB2 pureScale 集群+HADR 架构下，对现有的 pureScale 集群的连接方式做了增强，加上备集群的信息后，能够达到自动切换路由(ACR)的功能。在 DB2 V10.5 里，客户端主要有三种连接方式：

- WLB：工作负载均衡是现有的应用方式之一，能够根据成员上资源的消耗来安排新的工作到空闲的成员上去做，从而达到资源的有效利用。在引入了 HADR 功能之后，WLB 依然适用。只要配置好 Alternate Server 的信息指向备集群，在 HADR 切换之后，客户端会自动连接到新的主集群，WLB 也能正常生效。

- Client Affinity：在这种方式下，DB2 的客户端不需要获取所连接集群的信息，只需要按照自己的偏好列表里面定义的策略去访问节点。所以在偏好列表里面加上备集群的成员节点信息，就能够做到应用的自动切换。
- Member Subsetting：这是从 DB2 V10.5 开始加入的一种新的连接方式。在这种方式下，用户能够控制应用可以跑在指定的成员节点上。在这些成员节点上，应用还可以采用 WLB、Affinity 或 Round Robin 负载方式。对于 HADR 来说，这些定义都存储在数据库内部，所以切换之后，配置信息不会丢，应用还会延续先前的连接方式。

7.6.6　DB2 集群异地容灾架构的高可用性

DB2 pureScale 集群最大的特点就是具有高可用性、高扩展性和应用透明性。那么加入 HADR 的容灾机制之后，这些特性并没有受到影响。一方面应用连接数据库所使用的原有机制并没有改变。一旦集群里的节点出现故障，原有的集群保护机制依旧生效。那么对于 HADR 的工作机制来说，这种高可用性也是延续的。

主集群一旦成员节点发生故障，那么该节点的数据库服务会被其他节点所接管。HADR 服务也一样，其他健康的成员节点会帮助传输日志到备集群(日志在共享存储上)，从而达到高可用性。

备集群如果发生节点故障，一方面备集群的自我恢复能力和普通 pureScale 集群是一样的。另一方面，如果发生的故障正好影响到 HADR 功能，例如 Replay Member 被关机，那么 HADR 的功能也是会自我恢复的。例如选择一个健康的成员成为新的 Replay Member。

所以 pureScale 集群里的 HADR 功能同样也是高可用的。

7.6.7　DB2 集群异地容灾特性

DB2 pureScale 集群通过 HADR 功能，传输日志到备端。这种方式高效而安全，也延续了普通单机版 DB2 久经验证的 HADR 机制。只有生成日志的事务才需要被同步到备集群。HADR 的 spooling 功能能够在 Replay Member 来不及回放的情况下将日志先保存到磁盘，避免了高峰期对性能的影响且保证了数据的安全。

7.7　本章小结

DB2 集群突出的高可用性和易扩展性非常适合对可用性和性能要求很高的 OLTP 场景；但是 DB2 集群环境比较复杂，外部整合了 TSA 集群和 GPFS 集群，内部各个功能机制变化也很大。所以数据库管理员需要非常了解这套集群的技术细节，才能管理好集群。

第 8 章

DB2 高级监控

通过学习本系列书中的《循序渐进 DB2(第 3 版)》，读者可对 DB2 数据库中的快照监控、db2pd 和事件监控有个基本的了解和案例熟悉，在本章将会继续对 DB2 监控进行研究，使读者深入了解一些高级监控案例。

本章主要讲解如下内容：

- 利用表函数监控
- 监控指标和案例
- db2pd 及监控案例
- 事件监视器及监控案例
- db2mtrk 及监控案例
- 性能监控总结

8.1　利用表函数监控

DB2 数据库提供了很多表函数，通过这些表函数可以获得很多与数据库性能有关的信息，所以也可以通过使用这些表函数代替 GET SNAPSHOT 命令来收集这些监控数据。

快照表函数能够捕获的许多监视器元素都受控于监视器开关。如果快照表函数中的某些函数描述中提到特定监视器开关，就表明受控于该开关。

我们可以通过引用 20 多个可用的快照监视器表函数中的一个来构造 SQL 查询以收集

快照数据。表 8-1 列出了这些表函数并指明它们所能获取的具体快照信息。

表 8-1 部分表函数列表

快照表函数	返回的信息
MON_GET_INSTANCE	数据库管理器信息
MON_GET_AGENT	返回代理程序信息
MON_GET_CONTAINER	返回表空间容器信息
MON_GET_TABLESPACE	返回表空间的信息
MON_GET_DATABASE	数据库信息。仅当至少有一个应用程序连接至数据库时，才会返回信息
MON_GET_CONNECTION	连接至分区上数据库的应用程序上的有关锁等待的应用程序信息，包括累积计数器、状态信息和最近执行的 SQL 语句(假定设置了语句监视器开关)
MON_GET_UNIT_OF_WORK	每个连接至分区上数据库的应用程序的常规应用程序标识信息
MON_GET_APPL_LOCKWAIT	有关锁等待连接至分区上数据库的应用程序的应用程序信息
MON_GET_ACTIVITY	有关连接至分区上数据库的应用程序的语句的应用程序信息，包括最近执行的 SQL 语句(假定设置了语句监视器开关)
MON_GET_TABLE	连接至数据库的应用程序所访问的每个表的表活动信息，需要表监视器开关
MON_GET_LOCKS	数据库级别上的锁信息，以及每个连接至数据库的应用程序在应用程序级别上的锁信息。需要锁监视器开关
MON_GET_BUFFERPOOL	指定数据库的缓冲池活动计数器。需要缓冲池监视器开关
MON_GET_PKG_CACHE_STMT	来自用于数据库的 SQL 语句高速缓存的某个时间点的语句信息

可以通过下面的 SQL 语句返回所有的表函数：

```
    db2 "select distinct funcname from syscat.functions where funcname like
'MON_GET%'"
     FUNCNAME
    ----------------------------------------------------------------------
    MON_GET_ACTIVITY
    MON_GET_ACTIVITY_DETAILS
    MON_GET_AGENT
    MON_GET_APPLICATION_HANDLE
```

```
MON_GET_APPLICATION_ID
MON_GET_APPL_LOCKWAIT
MON_GET_AUTO_MAINT_QUEUE
MON_GET_AUTO_RUNSTATS_QUEUE
MON_GET_BUFFERPOOL
MON_GET_CF
MON_GET_CF_CMD
MON_GET_CF_WAIT_TIME
MON_GET_CONNECTION
MON_GET_CONNECTION_DETAILS
MON_GET_CONTAINER
MON_GET_DATABASE
MON_GET_DATABASE_DETAILS
MON_GET_EXTENDED_LATCH_WAIT
MON_GET_EXTENT_MOVEMENT_STATUS
MON_GET_FCM
MON_GET_FCM_CONNECTION_LIST
MON_GET_GROUP_BUFFERPOOL
MON_GET_HADR
MON_GET_INDEX
MON_GET_INDEX_USAGE_LIST
MON_GET_INSTANCE
MON_GET_LATCH
MON_GET_LOCKS
MON_GET_MEMORY_POOL
MON_GET_MEMORY_SET
MON_GET_PAGE_ACCESS_INFO
MON_GET_PKG_CACHE_STMT
MON_GET_PKG_CACHE_STMT_DETAILS
MON_GET_QUEUE_STATS
MON_GET_REBALANCE_STATUS
MON_GET_ROUTINE
MON_GET_ROUTINE_DETAILS
MON_GET_ROUTINE_EXEC_LIST
MON_GET_RTS_RQST
MON_GET_SECTION
MON_GET_SECTION_OBJECT
MON_GET_SECTION_ROUTINE
MON_GET_SERVERLIST
MON_GET_SERVICE_SUBCLASS
MON_GET_SERVICE_SUBCLASS_DETAILS
MON_GET_SERVICE_SUBCLASS_STATS
MON_GET_SERVICE_SUPERCLASS_STATS
```

```
    MON_GET_TABLE
    MON_GET_TABLESPACE
    MON_GET_TABLESPACE_QUIESCER
    MON_GET_TABLESPACE_RANGE
    MON_GET_TABLE_USAGE_LIST
    MON_GET_TRANSACTION_LOG
    MON_GET_UNIT_OF_WORK
    MON_GET_UNIT_OF_WORK_DETAILS
    MON_GET_USAGE_LIST_STATUS
    MON_GET_UTILITY
    MON_GET_WORKLOAD
    MON_GET_WORKLOAD_DETAILS
    MON_GET_WORKLOAD_STATS
    MON_GET_WORK_ACTION_SET_STATS

      61 record(s) selected.
```

快照监视器数据组织

所有的快照表函数都返回一张监视器数据表，其中的每一行代表一个正被监控的数据库对象实例，而每一列代表一个监视器元素。监视器元素代表数据库系统状态的特定属性。

捕获监视器数据快照

要使用快照表函数捕获直接访问的快照，请完成以下步骤：

(1) 连接至数据库。可以是需要监控的实例中的任何数据库。要能够使用快照表函数发出 SQL 查询，必须连接至数据库。

例如：

```
db2 connect to sample
```

(2) 确定需要捕获的快照类型，以及需要监控的数据库和分区。除了收集这些信息之外，请打开任何可应用的监视器开关(通过检查表 8-1 中的快照表函数可以确定这一点)。

例如，如果想要捕获表活动数据的快照(使用表函数 SNAPSHOT_TABLE)，那么将需要激活 TABLE 监视器开关：

```
db2 update dbm cfg using DFT_MON_TABLE on
```

(3) 使用期望的快照表函数发出查询。

例如，以下查询捕获有关当前已连接分区的 SAMPLE 数据库的表活动信息的快照：

```
db2 "select * from TABLE(MON_GET_TABLE('','',-2)) as T"
```

表函数有三个输入参数：

- VARCHAR(128)，用于 SCHEMA 名称。如果输入 NULL，就返回所有 SCHEMA 对应的表信息。VARCHAR(128)，用于表名称。如果输入 NULL，就返回所有表的信息。
- SMALLINT，用于分区号。对于分区号参数，输入整数(0 到 999 之间的值)以对应需要监控的分区号。要捕获当前已连接分区的快照，请输入值 -1 或 NULL。要捕获分区数据库所有分区的快照，请输入值 -2。

使用下面的语法将会创建引用非数据库管理器级表函数的查询：

```
SELECT*FROM TABLE(<FunctionName>(<DBName>,<PartitionNum>))AS CorrelationName
```

如果想要通过使用快照监视器的表函数 MON_GET_LOCKS 来抓取当前连接的数据库的锁定数据的信息，可以执行下面的语句：

```
SELECT * FROM TABLE (MON_GET_APPL_LOCKWAIT(NULL, -2)) AS LOCK_INFO
```

如果使用 SAMPLE 数据库(先前的例子)作为当前被连接的数据库，那么执行 GET SNAPSHOT FOR LOCKS ON SAMPLE 命令返回的信息将会与先前的表函数监控非常相似。

8.2　监控指标和案例

8.2.1　一些常用的监控指标和语句

下面介绍一些日常使用的案例，供大家参考。

1. 重组表和重组索引检查

在 DB2 数据库中，我们需要对表和索引定期地进行 reorg 操作，以使数据页能够连续地存储在表空间中，这样将大大提高数据库读 IO 的效率。在 DB2 中，我们可以使用如下两个函数获取当前系统中需要重组的表和索引信息，示例输出如下：

```
    $db2 -x "CALL SYSPROC.REORGCHK_TB_STATS('T', 'ALL')" | grep "\*" | awk
'{print "REORG TABLE " $1 "." $2 ";"}'
    REORG TABLE TEST.ADSPA_JX_TMP_1;
    REORG TABLE TEST.CZ_ADSPA_JX_TMP_1;
    REORG TABLE TEST.CZ_M_BP_PRIV_BASC_BASIC;
    REORG TABLE TEST.CZ_M_BP_PRIV_EXPA_ADDITIONAL;
    REORG TABLE TEST.CZ_M_BP_PRIV_EXPA_DOCUMENT;
    REORG TABLE TEST.CZ_M_BP_PRIV_EXPA_ELECADDR;
```

```
    REORG TABLE TEST.CZ M BP PRIV EXPA FIRST OPEN;
    REORG TABLE TEST.CZ_M_BP_PRIV_EXPA_LINKMAN;
```

在上面的操作命令中，使用 grep 输出带*的表信息，这些表都是需要进行 reorg 操作的表。通过后面的 awk 命令输出 reorg 语句，接下来我们就可以对这些表使用命令进行 reorg 操作。

同样，索引检查也可以这样进行操作，示例输出如下：

```
    $db2 -x "CALL SYSPROC.REORGCHK IX STATS('T', 'ALL')" | grep "\*" | awk
'{print $1 "." $2}'|more
    TEST.B DM DM END DAY BAL ADD OLD
    TEST.CZ M DEP PRIV FIXE ACCT INFO
    TEST.CZ M DEP PRIV FIXE ACCT INFO JXBF
    TEST.CZ M DEP PUB CURR ACCT INFO
    TEST.CZ M EUSP CM PAYBNM INFO
    TEST.CZ M EUSP DEP ESMCA
    TEST.CZ_M_EUSP_DEP_ESMCA_JXBF
```

接下来我们就可以根据这些信息对索引进行 reorg 操作。

2. 统计信息检查

在 DB2 数据库中，表的统计信息的准确是查询高效的保证，日常中我们需要定期或自动定时地对数据库中的表进行统计信息检查并进行统计信息更新。

在 DB2 中，我们可以通过 syscat.tables 表的 stats_time 字段距现在的时间来检查当前表有多久未进行统计信息操作。

在如下例子中，查询数据库当前 10 天未进行统计更新的表：

```
    select substr(TABSCHEMA,1,30) as TABSCHEMA,substr(TABNAME,1,50) as
TABNAME,STATS TIME from syscat.tables where TABSCHEMA not like 'SYS%' and
timestampdiff(16,char(current timestamp-timestamp(stats time)))>10;
    TABSCHEMA                      TABNAME
STATS TIME
    ------------------------------ ---------------- ----------------------------
    TEST                          M BP PRIV EXPA STREET H 20120920
2012-09-29-17.18.33.284932
    OGG                           TMP P3
2012-09-29-17.18.35.928153
    TEST                          CDSIA 20120715
2012-09-29-17.18.57.843692
    DB2DW                          AL LANG
2012-09-29-17.18.58.255853
```

```
    OGG                        TMP S47 TRANS
2012-09-29-17.18.58.860022
    OGG                        TMP S47 TRANS OGG
2012-09-29-17.18.59.923298
    TEST                       M GL SUBJ DAILY 20120412
2012-09-29-17.19.32.696310
    TEST                       FX PAPER GOLD 20120921
2012-09-29-17.19.52.962036
    TEST                       ISTL EMVCCTP 20120811
2012-09-29-17.19.55.252910
    TEST                       ISTL EXBLMSP 20120811
2012-09-29-17.19.55.601708
```

3. 缓冲池命中率

缓冲池是为加速数据查询而创建的，在数据库日常使用过程中，缓冲池会缓冲相当大一部分日常查询的数据到内存中，应用在访问数据库中将直接从内存中取到数据，快速返回结果集。但是如果缓冲池不包含需要查询的数据，这个时候就需要从磁盘读取数据到内存中，这会导致性能下降。

所以对缓冲池命中率的检查也是日常运维的重要工作，可以参考使用如下语句，从系统监控视图 **snapshot_bp** 中对缓冲池的数据缓冲命中率、索引缓冲命中率、缓冲池命中率、异步读命中率和直接读写命中率进行考量。

```
    select substr(bp name,1,20) as bp name, int ((1- (decimal(pool data p reads)
/ nullif(pool data l reads,0)))*100) as data hit ratio, int
((1-(decimal(pool index p reads)/nullif(pool index l reads,0)))*100) as
index hit ratio, int
((1-(decimal(pool data p reads+pool index p reads)/nullif((pool data l read
s+pool index l reads),0)))*100) as BP hit ratio, int
((1-(decimal(pool async data reads+pool async index reads)/nullif((pool asy

nc data reads+pool async index reads+direct reads),0)))*100) as
Async read pct, int ((1-(decimal(direct writes)/nullif(direct reads,0)))*100)
as Direct RW Ratio from table (snapshot bp ('dbname', -1)) as snapshot bp where
bp name not like 'IBMSYSTEM%';

    BP NAME              DATA HIT RATIO INDEX HIT RATIO BP HIT RATIO
ASYNC READ PCT DIRECT RW RATIO
    -------------------- -------------- --------------- ------------ ---------
    IBMDEFAULTBP                     99              99           99           99
100
```

```
    BP 32K                               95          99          98          97
94
```

我们需要定期检查缓冲池命中率，当命中率一直处于较低的值时，我们需要对缓冲池的大小进行调整。当然，在仓库系统中(OLAP)不用太刻意追求高的缓冲池命中率，因为仓库系统(OLAP)的数据交互太大，缓冲池不可能缓冲太多的数据。

4. 日志使用监控和长事务监控

在数据库的日常使用过程中，我们需要对日志的使用情况进行监控，当前日志的使用量和是否有长事务和大事务等，也都需要进行观察，否则日志占用满的话，整个数据库就无法继续运行下去了。

查看当前数据库的日志使用情况：

```
    select DBPARTITIONNUM,int(total log used/1024/1024) as "Log Used
(Mb)",int(total log available/1024/1024) as "Log Space Free(Mb)",
    int((float(total log used)/float(total log used+total log available))*1
00) as "Pct Used",int(tot log used top/1024/1024) as "Max Log Used (Mb)",
    int(sec log used top/1024/1024) as "Max Sec. Used
(Mb)",int(sec logs allocated) as "Secondaries" from sysibmadm.snapdb;

    DBPARTITIONNUM Log Used (Mb) Log Space Free(Mb) Pct Used    Max Log Used (Mb)
Max Sec. Used (Mb) Secondaries
    -------------- ------------- ------------------ ----------- --------------
                 2          46             15875           0            2863
0          0
                 1           3             15918           0            2881
0          0
                 3           3             15918           0            2893
0          0
                 4          79             15842           0            2883
0          0
                 6          60             15861           0            2862
0          0
                 7          26             15895           0            2878
0          0
                 5          20             15901           0            2927
0          0
                 8          19             15902           0            2993
0          0
                 0          44             15877           0             227
0          0
```

查看数据库中是否有长时间不提交的事务：

```
    select db.DBPARTITIONNUM,substr(ai.appl_status,1,20) as
"Status",substr(ai.primary_auth_id,1,10) as
"Authid",substr(ai.appl_name,1,15) as "Appl Name", int(ap
    .UOW_LOG_SPACE_USED/1024/1024) as "Log Used (M)",int(ap.appl_idle_time/60)
as "Idle for(min)",ap.appl_con_time as "Connected Since" from sysibmadm.snapdb
db,
    sysibmadm.snapappl ap,sysibmadm.snapappl_info ai where
ai.agent_id=db.APPL_ID_OLDEST_XACT and ap.agent_id=ai.agent_id;

    DBPARTITIONNUM Status               Authid     Appl Name       Log Used (M)
Idle for(min) Connected Since
    -------------- -------------------- ---------- --------------- -----------
                 7 UOWWAIT              DWUSER     al_engine                 0
0 2013-01-30-09.09.33.856817
                 7 UOWWAIT              DWUSER     al_engine                 0
0 2013-01-30-09.09.29.729616
                 7 UOWWAIT              DWUSER     al_engine                 0
0 2013-01-30-09.09.33.810848
                 7 UOWWAIT              DWUSER     al_engine                 0
0 2013-01-30-09.09.33.875796
                 7 UOWWAIT              DWUSER     al_engine                 0
0 2013-01-30-09.09.33.837672
                 7 UOWWAIT              DWUSER     al_engine                 0
    略...
```

查看数据库中消耗日志最大的 10 个事务：

```
    select DBPARTITIONNUM,AGENT_ID,int(UOW_LOG_SPACE_USED/1024/1024) as
Log_Used_MB from sysibmadm.snapappl where UOW_LOG_SPACE_USED<>0 order by
Log_Used_MB desc fetch first 10 rows only;
    DBPARTITIONNUM        AGENT_ID        LOG_USED_MB
    -------------- -------------------- -----------
                 6                 2296          10
                 6                80472           5
                 6               393676           0
                 6               393663           0
                 6                 8492           0
                 6                 2262           0
                 6               393670           0
                 6               393664           0
                 6                 8730           0
                 6               393671           0
```

5. 数据库对象状态的检查

另外，日常中较为重要的一项工作就是检查数据库对象的状态是否正常，以下是几种检查案例，供大家参考。

检查表状态

```
    select substr(tabschema,1,10) as tabschema,substr(tabname,1,30) as
tabname,status from syscat.tables where status != 'N' with ur;
```

检查处于 load pending 状态的表

有些时候，因为种种原因装载数据异常终止，我们可以使用如下语句查询数据库中处于 load pending 状态的表：

```
    select distinct substr(TABSCHEMA,1,30) as schema,substr(TABNAME,1,50) as
tabname,load status from SYSIBMADM.ADMINTABINFO where
load_status='PENDING'with ur;
```

如果有表处于 load pending 状态的话，可以使用如下命令将表恢复到正常状态：

```
    db2 "load from /dev/null of del terminate into schema.tabname"
```

检查 package 状态

```
    select substr(PKGSCHEMA,1,10) as PKGSCHEMA,substr(PKGNAME,1,30) as
PKGNAME,VALID from syscat.PACKAGES  where VALID !='Y' with ur;
    PKGSCHEMA  PKGNAME                         VALID
    ---------- ------------------------------ -----
    DWUSER    P9402730                       N
    NULLID     STADMG04                       X
    DB2DW     P7211730                       N
```

当 package 出现无效时，使用如下命令进行重新绑定：

```
    $db2 "rebind package DWUSER.P9402730 resolve any"
    DB20000I  The REBIND PACKAGE command completed successfully.
```

6. 数据库中热点表的检查

数据库中的热点表也需要我们定期加以关注，并对这些表进行调整，如下语句可以查出当前系统前 10 张热点表，并展示出当前表的读写信息，以及是否有 overflow_accesses 等信息。如果表的 overflow_accesses 很多的话，就需要对这个表进行 reorg 操作以降低跨页访问。

```
    select substr(table schema,1,20) as tbschema,substr(table name,1,30) as
tbname, rows read, rows written, overflow accesses, page reorgs from table
(SNAPSHOT TABLE(' ', -1)) as snapshot table order by rows written desc fetch
first 10 rows only;
    TBSCHEMA              TBNAME                            ROWS READ
ROWS WRITTEN          OVERFLOW ACCESSES    PAGE REORGS
-------------------- ------------------------------ -------------------- ----
    DWUSER               D CURE MA MONTHLY DATA                 88869489
47048553               0                   0
    DWUSER               M BP SA PE CUSTATTACHING                42106044
25311258               0                   0
    DWUSER               M BP SA CUST POINT SITU                 15408448
23097433               0                   0
    DWUSER               M BP SA VIP CUST INFO                  68605883
12151165               0                   0
    DWUSER               ADSPA JX TMP 1                        21196183
10823544               0               20373
    DWUSER               M EUSP PAYCHN ADSVA TMP 1               13022844
10569688               0               19676
    DWUSER               M EUSP DEP CDTJ ACCT CURR BALA           78463107
9864143               0               4468
    DWUSER               M NIN FDM F ACCT VCHR 1                324875909
7925113               0                   0
    DWUSER               M NIN MPAY ACCOUNT INFO                 15037720
6770118               0                   0
    DWUSER               TEMP M EUSP PAYCHN ADSPA INFO            13837463
6590861               0               4521
```

8.2.2　编写脚本以获取监控信息

上面已经介绍了很多监控指标和案例，我们这里介绍通过 shell 脚本对数据库进行监控以获取相应信息的方法。在日常数据库监控中大家可以根据需要对脚本进行调整。

当前脚本实现的功能为：查看当前服务器上所有处于活动状态的数据库，并使用 GET_DBSIZE_INFO 函数获取数据库的大小信息。

脚本如下：

```
    #!/bin/sh
    dir=/tmp/db2size
    mkdir -p /tmp/db2size

    test -s /usr/local/bin/db2ls
    if [ $? -eq 0 ]; then
```

```
           rm -f ${dir}/*.list ;
           rm -f ${dir}/*.db2
           ps -ef|grep db2sysc |grep -v grep |awk '{print $1}' >
${dir}/instance.list;
           cat ${dir}/instance.list |while read instance
           do
             su - ${instance} -c "db2 list db directory" |grep -p Indirect| grep
"Database alias" |awk -F'=' '{print $2}' > ${dir}/db.list
             cat ${dir}/db.list|while read db
             do
             echo "connect to $db;" >> ${dir}/$instance.db2
             echo "CALL GET DBSIZE INFO(?, ?, ?, -1);" >> ${dir}/$instance.db2
             done
             echo "---------------------------------------------"
             banner ${instance}
             echo " "
             echo "------------"
             su - ${instance} -c "db2 -tvf ${dir}/$instance.db2 "
           done
    fi
```

输出信息如下：

```
    ---------------------------------------------
                   ######
                   ######
     ######  ###### #     #  ######  #   #
     #    #  #   #       # #    #  #    #
     #    #  ######  ######  #   #  #    #
     #    #  #   # #        #   #  # ## #
     #    #  #   # #        #   #  ##  ##
     ######  ###### ######## ######   #   #
    ------------
    connect to DASDB

       Database Connection Information

     Database server        = DB2/AIX64 9.7.5
     SQL authorization ID   = DB2MDSC
     Local database alias   = DASDB


    CALL GET DBSIZE INFO(?, ?, ?, -1)
```

```
Value of output parameters
--------------------------
Parameter Name  : SNAPSHOTTIMESTAMP
Parameter Value : 2013-01-30-10.31.08.450992

Parameter Name  : DATABASESIZE
Parameter Value : 27256193024

Parameter Name  : DATABASECAPACITY
Parameter Value : 138191466496

Return Status = 0
```

8.3　db2pd 及监控案例

8.3.1　db2pd 概述

db2pd 是用于监视各种 DB2 数据库活动以及故障排除的监控工具。它是从 DB2 V8.2 开始随 DB2 引擎发布的一款独立的实用程序，其外观和功能类似于 Informix onstat 实用程序(其实就是 IBM 收购 Informix 后，DB2 从 Informix 数据库那里借鉴过来的)。db2pd 是从命令行以一种可选的交互模式执行的。该实用程序运行得非常快，因为它不需要获取任何锁，并且在引擎资源以外运行(这意味着它甚至能在挂起的引擎上工作)。通过快照监视还可以收集 db2pd 提供的很多监视器数据，但 db2pd 和快照监视的输出格式有很大不同。

下面我们将以更多例子向大家展示 db2pd 在 DB2 监控中的强大特性，使大家对 db2pd 有更深了解。

8.3.2　db2pd 监控案例

1. 锁等待分析场景

用户 A 执行事务 A，以根据每个经理的薪水为他们提供 10%的奖金。

事务 A 执行的更新操作：

```
UPDATE EMPLOYEE SET BONUS = SALARY * 0.1  WHERE JOB = 'MANAGER'
```

当事务 A 仍然在运行时(因为用户 A 还没有使用 COMMIT 或 ROLLBACK 终止该事务)，用户 B 执行事务 B，以将每个雇员的薪水提高 2%。

事务 B 执行的更新操作：

```
UPDATE EMPLOYEE SET SALARY = SALARY * 0.02
```

由于事务 B 没有完成，用户 B 请求 DBA 确定问题的原因。于是，DBA 调用 db2pd，看是否存在锁等待情形。

检查锁等待情形：

```
db2pd -db sample -locks wait showlocks
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:33:05
Locks:
Address    TranHdl    Lockname             Type       Mode Sts Owner      Dur
0x050A0240 6      02000600050040010000000052 Row      ..X  W   2          1
0x050A0DB0 2      02000600050040010000000052 Row      ..X  G   2          1
HoldCount  Att  ReleaseFlg
0  0x00 0x40000000   TbspaceID 2  TableID 6  PartitionID 0 Page 320 Slot 5
0  0x00 0x40000000   TbspaceID 2  TableID 6  PartitionID 0 Page 320 Slot 5
```

db2pd 报告 ID 为 2 的表空间中 ID 为 6 的表上有行锁存在锁等待情形。通过检查 SYSCAT. TABLES，DBA 断定表 EMPLOYEE 上的确存在锁等待情形。

确定锁等待情形涉及的表：

```
 SELECT TABSCHEMA, TABNAME FROM SYSCAT.TABLES
WHERE TBSPACEID = 2 AND TABLEID = 6
TABSCHEMA                                 TABNAME
-----------------------------------------------------------------------
ORACLE                                 EMPLOYEE
  1 record(s) selected.
```

对于事务 2(列 TranHdl)，db2pd -locks 输出的 status 列(Sts)显示“G”。G 代表“granted”，意思就是事务句柄为 2 的事务拥有行锁。此外，列 Mode 表明事务 2 持有的是 X 锁。等待的事务(列 Sts 中显示“W”(“wait”)的事务)是句柄为 6 的事务。该事务正在与事务 2 请求同一行上的 X 锁。通过查看 Owner 列(显示事务 2 是锁的所有者)和比较 Lockname(对于 db2pd -locks 中的两个条目是相同的)，可以看到这一点。

接下来，DBA 将事务句柄映射到应用程序。这可以使用另一个 db2pd 选项-transactions 来完成。

将事务句柄映射到应用程序：

```
db2pd -db sample -transactions
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:34:47
Transactions:
Address    AppHandl [nod-index] TranHdl  Locks   State   Tflag      Tflag2
0x05141880 30      [000-00030]  2        9       WRITE   0x00000000 0x00000
```

```
0x05144880 34     [000-00034]   6       5        WRITE   0x00000000 0x00000
```

这个 db2pd 调用的输出表明，事务 2(列 TranHdl)是由应用程序 30(列 AppHandl)执行的，而事务 6 是由应用程序 34 执行的。这两个事务正在对数据库执行写更改(列 State = WRITE)，所以 DBA 现在知道，应用程序 30 正持有应用程序 34 所等待的锁。

要获得关于锁等待情形涉及的应用程序的更多信息，可使用-agents 选项调用 db2pd。该选项打印代表应用程序运行的代理的信息。注意，-agents 是实例级选项，这意味着不需要指定数据库(实际上，当指定数据库时，db2pd 打印出一条警告并忽略 database 选项)。

获得关于应用程序和相应代理的信息：

```
db2pd -agents
Database Partition 0 -- Active -- Up 3 days 08:35:42
Agents:
Current agents:      2
Idle agents:        0
Active coord agents: 2
Active agents total: 2
Pooled coord agents: 0
Pooled agents total: 0
Address    AppHandl [nod-index] AgentTid  Priority  Type    State
0x04449BC0 34       [000-00034] 3392      0         Coord   Inst-Active
0x04449240 30       [000-00030] 2576      0         Coord   Inst-Active
ClientPid  Userid   ClientNm Rowsread   Rowswrtn   LkTmOt DBName
3916       USER_B   db2bp.ex   43          43        NotSet SAMPLE
2524       USER_A   db2bp.ex   153        14        NotSet SAMPLE
```

在 db2pd –agents 的输出中，DBA 可以看到使用应用程序 30 和 34 的用户的 ID(列 Userid)：应用程序 30 是由 USER_A 执行的，而应用程序 34 是由 USER_B 执行的。只有当每个用户都有单独的数据库授权 ID 时，才可能出现那样的应用程序与用户 ID 之间的映射。通常，这对于在应用服务器上运行的应用程序是不可能的，因为这些应用程序使用连接池，连接不是个人化的。

关于每个应用程序的更多信息则由 db2pd 选项-applications 提供。

获得关于应用程序的更多信息：

```
db2pd -db sample -applications
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:36:14
Applications:
Address    AppHandl [nod-index] NumAgents  CoorTid    Status
0x04AF8080 34       [000-00024] 1          3940       Lock-wait
0x03841960 30       [000-00020] 1          2548       UOW-Waiting
```

```
C-AnchID C-StmtUID  L-AnchID L-StmtUID  Appid
195      1          0        0          *LOCAL.DB2.061122195637
0        0          60       1          *LOCAL.DB2.061122195609
```

Status 列确认了 DBA 已经知道的一些东西：应用程序 34 处于锁等待状态。但是这并不新鲜，于是 DBA 将注意力集中在列 C-AnchID/C-StmtUID 和 L-AnchID/L-StmtUID 上。“C”代表当前(current)，“L”代表最近(last)的锚 ID/语句 UID。这些 ID 可用于标识应用程序最近执行的 SQL 语句和应用程序当前执行的语句。为此，可以用-dynamic 选项调用 db2pd。该选项显示数据库动态语句缓存的内容。

2. 检查动态语句缓存的内容

```
   db2pd -db sample -dynamic
Database Partition 0 -- Database SAMPLE -- Active -- Up 3 days 08:37:39
Dynamic Cache:
Current Memory Used          187188
Total Heap Size             1271398
Cache Overflow Flag          0
Number of References         2
Number of Statement Inserts   3
Number of Statement Deletes   0
Number of Variation Inserts   2
Number of Statements         3
Dynamic SQL Statements:
Address    AnchID StmtUID   NumEnv    NumVar    NumRef    NumExe
0x056CEBD0 60     1         1         1         1         1
0x056CE850 180    1         0         0         0         0
0x056CFEA0 195    1         1         1         1         1
Text
UPDATE EMPLOYEE SET BONUS = SALARY * 0.1 WHERE JOB = 'MANAGER'
SET CURRENT LOCALE LC CTYPE = 'de DE'
UPDATE EMPLOYEE SET SALARY = SALARY * 0.02
Dynamic SQL Environments:
Address    AnchID StmtUID   EnvID Iso QOpt Blk
0x056CECD0 60     1         1     CS  5    B
0x056D30A0 195    1         1     CS  5    B
Dynamic SQL Variations:
Address    AnchID StmtUID   EnvID VarID     NumRef    Typ
0x056CEEB0 60     1         1     1         1         4
0x056D3220 195    1         1     1         1         4
Lockname
010000000100000001003C0056
```

```
0100000001000000100C30056
```

-applications 输出与-dynamic 输出之间的映射很简单：应用程序 34(处于锁等待状态)当前正在执行当前锚 ID195 和当前语句 ID1 所标识的 SQL 语句。在 db2pd -dynamic 输出的 Dynamic SQL Statements 部分，那些 ID 可以映射到以下 SQL 语句。

应用程序 34 执行的 SQL 语句：

```
UPDATE EMPLOYEE SET SALARY = SALARY * 0.02
```

持有锁的应用程序 30 最近执行的 SQL 语句是最近锚 ID60 和最近语句 ID1 所标识的 SQL 语句。那些 ID 可以映射到以下 SQL 语句。

应用程序 30 执行的 SQL 语句：

```
UPDATE EMPLOYEE SET BONUS = SALARY * 0.1 WHERE JOB = 'MANAGER'
```

注意，db2pd -dynamic 输出包含另一条通常难以发现的有趣信息：Dynamic SQL Environments 部分的列 Iso 显示了被执行的动态 SQL 语句的隔离级别(UR = Uncommitted Read，CS = Cursor Stability，RS = Read Stability，RR = Repeatable Read)。

我们总结一下 DBA 就用户 B 的应用程序被挂起的原因有什么发现：

- 挂起是由表 EMPLOYEE 上某个独占式的行锁导致的。
- 持有锁的事务属于用户 A 执行的某个应用程序，而用户 B 的事务正在等待那个锁。
- 两条有冲突的语句都是表 EMPLOYEE 上的 UPDATE 语句。

有了这些信息，DBA 就可以开始采取一些必要的步骤来解决锁等待状况，例如建议用户 A 终止事务，或者强制关闭用户 A 的应用程序。此外，可以采取措施避免将来出现那样的状况，例如检查是否创建了最合理的索引，使之通过索引扫描快速提交事务并释放锁。

在这个示例场景中，db2pd 被连续执行数次，每次使用单独的选项。现实中不会出现这样的情况。相反，db2pd 只被调用一次，调用时同时使用前面介绍的所有选项。

分析锁等待情形所需的带有所有选项的单个 db2pd 调用：

```
    db2pd -db sample -locks wait showlocks -transactions -agents -applications
-dynamic     -file db2pd.out -repeat 15 40
```

产生的输出由针对每个选项的输出组成，各部分输出之间的顺序与各选项在 db2pd 调用中的顺序一致。而且，请注意 db2pd 调用最后的两个附加选项：

- -file 表明 db2pd 输出应该被写到文件中。在示例调用中，输出被写到文件 db2pd.out 中。
- -repeat 表明 db2pd 应该每隔 15 秒执行一次，共执行 40 次(每隔 15 秒执行一次，共执行 10 分钟)。每次执行的输出被附加到-file 选项指定的文件后面。

-file 和-repeat 选项对于在一段时间内监视数据库活动比较有用。对于锁等待分析，这

两个选项可以帮助捕捉只存在一小段时间的锁等待情形。例如，如果数据库参数 LOCKWAIT 被设置为 20 秒，那么等待锁的事务在过了 20 秒的等待时间后被回滚。为了捕捉那样的锁等待情形，db2pd 的时间间隔必须设置为比 20 秒更短的时间间隔，例如例子中的 15 秒。

3. 捕捉罕见的锁超时

有时候，锁等待会导致锁超时，而锁超时又会导致事务被回滚。锁等待导致锁超时所需的时间段由数据库配置参数 LOCKTIMEOUT 指定。锁超时分析最大的问题是，不知道下一次的锁超时何时发生。为了捕捉死锁，可以创建死锁事件监视器。每当出现死锁时，这个死锁事件监视器便写一个条目。但是，对于锁超时就没有类似的事件监视器。所以到 DB2 V9 为止，捕捉锁超时的唯一方法还是连续的 db2pd 或快照监视(对于 db2pd，和前面解释的一样，-file 和-repeat 选项可用于连续的锁监视)。

DB2 V9 包含了一种新的机制，用于在数据库出现故障或发生事件时收集监视器数据—— db2cos 脚本。为了捕捉锁超时事件，可以配置数据库，使之每当出现锁超时启动 db2cos 脚本。在 db2cos 脚本中，和前面讨论的一样，可以以相同的选项调用 db2pd。我们来看一个示例场景，该场景演示了如何用 db2cos 脚本捕捉锁超时。

对于这个场景，假设 DBA 将数据库锁超时设为 10 秒。

更新锁超时设置：

```
UPDATE DB CFG FOR SAMPLE USING LOCKTIMEOUT 10
```

为了每当出现锁超时时启动 db2cos 脚本，DBA 调用 db2pdcfg 实用程序。

使用 db2pdcfg 配置 db2cos 脚本的调用：

```
db2pdcfg -catch locktimeout count=1
```

-catch 选项指定应该自动导致调用 db2cos 脚本的故障或事件。对于锁超时事件，可以指定字符串 locktimeout。或者，可以指定与锁超时相应的 SQL 错误码和原因码。

用于捕捉锁超时的另一种 db2pdcfg 调用：

```
db2pdcfg -catch 911,68 count=1
```

除了一些字符串值和 SQL 代码之外，db2pdcfg 还接受内部 DB2 错误码。所以，用这种方式可以捕捉很多数据库故障和事件。锁超时事件只是使用 db2pdcfg 和 db2cos 的一种情况。

如果 count 子选项的值为 1，那么表明当出现锁超时事件时应该执行 db2cos 脚本。

db2pdcfg 通过以下输出确认错误捕捉的设置：

```
Error Catch #1
Sqlcode:          911
ReasonCode:       68
ZRC:             -2146435004
ECF:              0
Component ID:     0
LockName:         Not Set
LockType:         Not Set
Current Count:    0
Max Count:        1
Bitmap:          0x4A1
    Action:          Error code catch flag enabled
Action:         Execute sqllib/db2cos callout script

Action:         Produce stack trace in db2diag.log
```

db2diag.log 报告中也包括错误捕捉设置。可以使用 db2diag 实用程序(用于检查 db2diag.log 内容的实用程序)过滤 db2diag.log 文件，而不必在文本编辑器中打开。

在 db2diag.log 中确认错误捕捉设置：

```
db2diag -g funcname:=pdErrorCatch
2012-12-18-13.37.25.177000+060 I727480H285       LEVEL: Event
PID     : 4648                TID  : 3948       PROC : db2syscs.exe
INSTANCE:DB2                  NODE : 000
FUNCTION:DB2UDB, RAS/PD component, pdErrorCatch, probe:30
START   : Error catch set for ZRC -2146435004
```

ZRC-2146435004 是用于锁超时的 DB2 内部错误码。可以通过下面的 db2diag 调用查看这些错误码。

使用 db2diag 查看 DB2 内部错误码的含义：

```
db2diag -rc -2146435004
```

通过使用 db2pdcfg，数据库引擎现在被配置为每当出现锁超时时调用 db2cos 脚本。db2cos 脚本收集判别锁超时原因所需的所有监视器信息。为此，DBA 必须修改 db2cos 脚本，以使用已知的选项调用 db2pd。可以在下面的子目录中找到 db2cos 脚本：

- 在 Windows 环境下，DB2 install directory\BIN\db2cos.bat，例如 C:\Program Files\IBM\ SQLLIB\BIN\db2cos.bat。
- 在 UNIX/Linux 环境下，Instance owner home/sqllib/bin/db2cos。

在 Windows 环境下，默认的 db2cos.bat 脚本看上去如下所示：

```
setlocal
```

```
:iterargs
if %0. == . goto iterdone
if /i %0. == INSTANCE. set INSTANCE=%1
if /i %0. == DATABASE. set DATABASE=%1
      if /i %0. == TIMESTAMP. set TIMESTAMP=%1
 if /i %0. == APPID. set APPID=%1
       if /i %0. == PID. set PID=%1
       if /i %0. == TID. set TID=%1
       if /i %0. == DBPART. set DBPART=%1
if /i %0. == PROBE. set PROBE=%1
      if /i %0. == FUNCTION. set FUNCTION=%1
      if /i %0. == REASON. set REASON=%1
      if /i %0. == DESCRIPTION. set DESCRIPTION=%1
      if /i %0. == DiAGPATH. set DIAGPATH=%1
 shift
goto iterargs
:iterdone
if %DATABASE%. == . goto no database
db2pd -db %DATABASE% -inst >> %DIAGPATH%\db2cos%PID%%TID%.%DBPART%
goto exit
:no database
 db2pd -inst >> %DIAGPATH%\db2cos%PID%%TID%.%DBPART%
:exit
```

对于数据库级的事件或故障，默认的 db2cos 脚本用-db 和-inst 选项调用 db2pd。DBA 用 db2pd 调用替换相应的行，该调用收集锁超时分析所需的监视器数据。

更改 db2cos 脚本以收集用于锁超时分析的数据：

```
if %DATABASE%. == . goto no database
db2pd -db %DATABASE% -locks wait -transactions -agents -applications -dynamic
   >> %DIAGPATH%\db2cos%PID%%TID%.%DBPART%
 goto exit
```

现在，db2cos 脚本已准备好，DBA 可以坐等下一次锁超时事件的发生。

假设像之前描述的那样，用户 A 与 B 之间发生相同的锁情形。但是，这一次设置了 LOCKTIMEOUT，因此过了 10 秒(LOCKTIMEOUT = 10)之后用户 B 的事务被回滚。用户 B 通知 DBA 回滚自己的事务，并且收到 SQL 错误消息-911 和原因码 68(SQL code -911 / reason code 68 = locktimeout)。于是，DBA 检查通过自动调用 db2cos 脚本收集到的监视器数据。

首先，DBA 用锁超时内部错误码调用 db2diag，进而确定锁超时发生的确切时间。

在 db2diag.log 中检查锁超时事件的时间点：

```
db2diag -g data:=-2146435004
2012-11-18-14.27.24.656000+060 I6857H409         LEVEL: Event
PID     : 2968                  TID  : 2932       PROC : db2syscs.exe
INSTANCE:DB2                   NODE : 000         DB   : SAMPLE
APPHDL  : 0-21                  APPID: *LOCAL.DB2.061226132544
AUTHID  : FECHNER
FUNCTION:DB2UDB, lock manager, sqlplnfd, probe:999
DATA #1 : <preformatted>
Caught rc -2146435004.  Dumping stack trace.
```

db2diag.log 条目显示，在 2012-11-18-14.27.24.656000 时发生了一次锁超时。由于 db2cos 脚本将输出写到%DIAGPATH%中的 db2cos%PID%%TID%.%DBPART% 文件中，DBA 有望在实例的诊断路径中找到 db2cos29682932.0 文件：

- %DIAGPATH% = instance's diagnostic path = on Windows by default C:\Program Files\IBM\SQLLIB\DB2
- %PID% = processID= 2968(如 db2diag.log 条目中所示)
- %TID% = threadID= 2932(也显示在 db2diag.log 条目中)
- %DBPART% = database partition = 0(在非分区数据库环境中)

文件的内容与前面逐步考查的那个 db2pd 监视器输出相似，DBA 可以借此识别锁超时的原因。

捕捉到锁超时后，DBA 可以通过-catch clear 选项调用 db2pdcfg 来禁用 db2cos 脚本。

再次使用 db2pdcfg 清除错误捕捉设置：

```
db2pdcfg -catch clear
All error catch flag settings cleared.
```

4. 查看并 force 掉数据库上某个表的所有连接

在生产环境中，有些时候当需要对某张表进行 load 操作或 drop 操作时，往往这个表上有连接未释放，导致不能操作。

在这种情况下，在生产中是绝对不允许执行 force application all 这种操作的，在如下案例中我们一步一步教大家如何找出表上的所有连接并杀掉。

假设我们需要对表 TEST. TRACKSTMTS 进行操作，但是目前表上有其他会话持有锁，我们无法获取表锁。

首先查找表的 TBSPACEID 和 TABLEID：

```
$db2 "select TBSPACEID,TABLEID from syscat.tables where TABSCHEMA='TEST'
and tabname='TRACKSTMTS'"
```

```
    TBSPACEID TABLEID
    --------- -------
            2    3334

      1 record(s) selected.
```

我们可以看到，当前表的 TBSPACEID 为 2、TABLEID 为 3334，得到这个有什么作用呢？在 db2pd –d dbname –lock showlock 中记录了当前数据库所有的锁定信息，而且在-lock showlock 输出的第 13、14、15、16 行就有对应表的 TBSPACEID、TABLEID 信息，我们只需要匹配出这些就可以找到表上所有的连接信息。

如下是一个简单的使用 shell 实现找到这些锁定信息的例子，供大家参考。在 DB2 V9.5 和 DB2 V9.7 中，TBSPACEID 和 TABLEID 在 db2pd –lock showlock 中的位置可能不同，需要大家确认修改。这里我们是以 DB2 V9.7 为例。

输出表上所有的连接：

```
    $db2pd -d dw -lock showlock|awk '{if($14=="2" && $16=="3334")print $0 }'

    0x07000001B8963100 6         00020D06000000000000000054 Table      .IX  G
6     1   1          0x00002000 0x40000000 0          TbspaceID 2     TableID 3334
    0x07000001B8963180 8         00020D06000000000000000054 Table      .IX  G
8     1   1          0x00002000 0x40000000 0          TbspaceID 2     TableID 3334
    0x07000001B8963D00 10        00020D06000000000000000054 Table      .IX  G
10    1   1          0x00002000 0x40000000 0          TbspaceID 2     TableID 3334
    0x07000001B8964900 9         00020D06000000000000000054 Table      .IX  G
9     1   1          0x00002000 0x40000000 0          TbspaceID 2     TableID 3334
    0x07000001B8966100 5         00020D06000000000000000054 Table      .IX  G
5     1   1          0x00002000 0x40000000 0          TbspaceID 2     TableID 3334
    0x07000001B8965500 11        00020D06000000000000000054 Table      .IX  G
11    1   1          0x00002000 0x40000000 0          TbspaceID 2     TableID 3334
    0x07000001B8980200 12        00020D06000000000000000054 Table      .IX  G
12    1   1          0x00002000 0x40000000 0          TbspaceID 2     TableID 3334
    0x07000001B8966C80 7         00020D06000000000000000054 Table      .IX  G
7     1   1          0x00002000 0x40000000 0          TbspaceID 2     TableID 3334
```

这样就可以确定数据库的连接一共是 8 个，我们可以获取当前的 TranHdl，进而去匹配并得到 AppHandl。

如下是匹配 AppHandl 的操作方法：

```
    $db2pd -d dw -tran |awk '{if($4=="6")print $0}'
    0x07000001B2455A80 458877   [007-00125] 6         1         READ
0x00000000 0x00000000 0x0000000000000000 0x0000000000000000 0            0
```

```
0x0000028AB4ED 1        n/a                   n/a                n/a               n/a
```

我们已经找到 TranHdl 6 对应的 AppHandl 为 458877，接下来就可以方便我们查看连接的具体执行语句信息了。如果需要 force 的话，可以执行如下操作：

```
 db2 "force application (458877)"
```

其他的获取过程与此相同，这里不再叙述。

5. 查看某个表上索引的扫描情况

在 DB2 数据库的使用过程中，大家有没有碰见过应用人员在创建索引时在表上创建了一大堆索引，甚至有的时候索引的大小都超过了数据的大小，真的有没有必要创建那么多的索引？作为运维人员很难去直接判断，这个时候我们就需要对索引扫描进行监控。

接下来的例子我们一起来看看索引扫描在数据库里面如何通过 db2pd 进行监控。

在这里我们以表 TEST. T87_ACC_INCOME_BAL 为例，首先需要监控表的 TBSPACEID 和 TABLEID：

```
    $db2 "select tbspaceid,tableid from syscat.tables where tabschema='TEST'
and tabname='T87 ACC INCOME BAL'"

    TBSPACEID TABLEID
    --------- -------
          22     155

      1 record(s) selected.
```

确定了 TBSPACEID 为 22、TABLEID 为 155，接下来通过 db2pd 监控我们就可以得到表上索引扫描的信息：

```
    FRPTDB01:/db2/db2frp$db2pd -d FRPDB -tcbstats index 22 155

    Database Partition 0 -- Database FRPDB -- Active -- Up 96 days 18:43:40 --
Date 01/30/2013 14:55:59

    TCB Table Information:
    Address          TbspaceID TableID PartID MasterTbs MasterTab TableName
SchemaNm ObjClass DataSize   LfSize     LobSize    XMLSize
    0x070000089D62F258 22       155     n/a    22        155
T87 ACC INCOME BAL TEST   Perm     615       0         0         0

    TCB Table Stats:
```

```
    Address            TableName          SchemaNm Scans      UDI        RTSUDI
PgReorgs   NoChgUpdts Reads      FscrUpdates Inserts    Updates    Deletes
OvFlReads  OvFlCrtes  RowsComp   RowsUncomp CCLogReads StoreBytes BytesSaved
    0x070000089D62F258 T87_ACC_INCOME_BAL TEST    55260      3441       3441
533406     40238      3372852925 191166      230297     1258165    151382
1460595482 91172      1435067    3265975013 613503     -          -

    TCB Index Information:
    Address            InxTbspace ObjectID PartID TbspaceID TableID MasterTbs
MasterTab TableName          SchemaNm IID   IndexObjSize
    0x0700000576B90CD0 23         1301     n/a    22        155     22        155
T87_ACC_INCOME_BAL TEST   3     2414
    0x0700000576B90CD0 23         1301     n/a    22        155     22        155
T87_ACC_INCOME_BAL TEST   2     2414
    0x0700000576B90CD0 23         1301     n/a    22        155     22        155
T87_ACC_INCOME_BAL TEST   1     2414

    TCB Index Stats:
    Address            TableName          IID   PartID EmpPgDel   RootSplits
BndrySplts PseuEmptPg EmPgMkdUsd Scans      IxOnlyScns KeyUpdates InclUpdats
NonBndSpts PgAllocs   Merges     PseuDels   DelClean   IntNodSpl
    0x0700000576B90CD0 T87_ACC_INCOME_BAL 3     n/a    0          0          0
0          0          1882       1          0          0          22         362
0          10         9          0
    0x0700000576B90CD0 T87_ACC_INCOME_BAL 2     n/a    0          0          0
0          0          1449       1024       236835     0          612        1239
0          236713     208970     10
    0x0700000576B90CD0 T87_ACC_INCOME_BAL 1     n/a    0          0          92
0          0          267229     4055       0          0          0          811
0          10         4          1
```

从上面的 Scans 项，我们可以看到表上索引的扫描情况和对应的 IID 信息。如果某个索引扫描次数是 0 的话，可以使用如下语句找到索引名并确定是否删除索引：

```
    select SUBSTR(TABSCHEMA,1,20) AS TABSCHEMA,SUBSTR(TABNAME,1,20) AS
TBNAME,IID,SUBSTR(INDNAME,1,15) AS INDEX_NAME,SUBSTR(COLNAMES, 1, 50) AS
COLUMNS from syscat.indexes where TABSCHEMA='TEST' and TABNAME='T87_ACC_INCOME_BAL'
order by iid DESC
    TABSCHEMA            TBNAME               IID    INDEX_NAME      COLUMNS
-------------------- -------------------- ------ ------------------------------
    TEST              T87_ACC_INCOME_BAL       3 I_T87_ACC      +ACCOUNTNO
    TEST              T87_ACC_INCOME_BAL       2 I_T87_ACC_IB
+ACCOUNTNO+ORGANKEY+PATCH_DESC+CHECK_STAT+SEND_STA
```

```
TEST           T87 ACC INCOME BAL      1 P PK T87 FEX IN +RECORDNUM

  3 record(s) selected.
```

8.4　事件监视器及监控案例

快照监视器监控的是数据库的实时数据，Event Monitor(事件监视器)记录某事件(event)或转变(transition)出现时某段时间内的数据库活动情况。事件监视器收集监视器数据，例如特定事件或事务发生。因此，事件监视器提供了当事件或活动发生的时候，不能使用快照监视器监视时收集数据库系统监视器数据的方法。例如，要捕获每当死锁周期发生时的监视器数据。如果对死锁的概念比较熟悉的话，就应该知道数据库拥有被称为死锁检测器的特殊进程(db2dlock)，它在后台安静地运行，并且在预定的间隔(dlchktime)时间内会“苏醒”，用于为死锁周期扫描当前正在锁定的系统。如果在死锁周期内被发现，死锁检测器将会随机选择、回滚并且终止涉及在此次周期内的任意事务。结果，被选择出来的那个事务将会接收到 SQL 错误代码(-911)，并且所有实际上已经获得的锁被释放，以便剩下的事务能够继续执行。像这样的一系列事件的信息将不能被快照监视器捕获，当死锁出现时，DB2 通过对多个事务中的某个事务发出 ROLLBACK(回退)操作去解决死锁问题。有关死锁操作的信息也不容易用 Snapshot Monitor 捕获到，因为在能够拍到快照之前，死锁可能已经被解决了。然后，事件监视器却可以捕获该事件的重要信息，因为它可以在死锁周期被检测到的瞬间被激活。这两种监视器的另外一个显著的不同是：快照监视器以后台进程的方式驻留，从数据库连接建立就开始捕获监视器数据；相反，事件监视器必须在它们被使用之前专门去建立和激活。几个不同的事件监视器可以共存，并且每个事件监视器只有在特定类型的事件或事务发生的时候才会被激活。

Event Monitor 类似其他的数据库对象，因为它们是使用 SQL DDL(数据定义语言)创建的。主要差别是 Event Monitor 可以像 Snapshot Monitor 的开关那样被打开或关闭。

当创建 Event Monitor 时，必须声明要被监视事件的类型。当以下事件发生时，相应事件记录便被记录下来：

- DATABASE——当最后一个应用程序与数据库断开时，记录下一个事件记录。
- TABLES——当最后一个应用程序与数据库断开时，记录每个活动表的事件记录。
- DEADLOCKS——对于每个死锁，记录事件记录。
- TABLESPACES——当最后一个应用程序与数据库断开时，对每个活动表空间记录事件记录。

- CONNECTIONS——当应用程序与数据库断开时，对每个数据库连接事件记录事件记录。
- STATEMENTS——对于由应用程序发出的每条 SQL 语句(动态或静态)，记录事件记录。
- TRANSACTIONS——当事务完成(COMMIT 或 ROLLBACK)时，为该事务记录事件记录。

Event Monitor 的输出存放在目录或命名管道(pipe)中(从 DB2 V8 后可以存放在表中，这些事件监控的表可以自动生成)。当 Event Monitor 被激活时，管道和文件的存在将得到证实。如果 Event Monitor 的目标位置是命名管道，那么提示从管道读出数据是应用程序的职责。如果 Event Monitor 的目标位置是目录，数据流将被写入一系列文件中。这些文件顺序编号并且都有文件附加名 evt(例如 00000000.evt、00000001.evt 等)。当定义事件监视器时，规定 Event Monitor Event 文件的大小与数目。

8.4.1 事件监视器的创建方法和步骤

(1) 创建 SQL Event Monitor，写入文件。语法如下：

```
db2 create event monitor evmname for eventtype write to file 'directory'
```

例如：

```
db2 create event monitor SQLCOST for deadlocks,statements write to file
'/db2db/event'
```

(2) 激活事件监视器(确保有充足的可用磁盘空间)：

```
db2 "set event monitor SQLCOST state = 1"
```

(3) 让应用程序运行。

(4) 取消激活事件监视器：

```
db2 "set event monitor SQLCOST state = 0"
```

(5) 使用 DB2 提供的 db2evmon 工具格式化 SQL Event Monitor 原始数据(根据 SQL 吞吐率可能需要数百兆字节的可用磁盘空间)：

```
$> db2evmon -db DBNAME -evm SQLCOST > sqltrace.txt
```

(6) 浏览整个已格式化的文件，寻找显著大的成本数(这是耗时的过程)：

```
$> more sqltrace.txt
```

其实在实际的性能调优中，事件监视器并没有被广泛运用，这是因为过去事件监视器的输出只能写到文件或命名管道中，直到 DB2 V8 才可以写到表中，使得 DBA 可以利用 SQL 去读取；其次是因为 DB2 的事件监视器在监控期间会产生非常大的文件。

8.4.2　事件监控器案例

下面我们举一个事件监视器存放在表中的例子。

对于数据库管理员，调优数据库常常是一项挑战。调优应用程序是一种方法，但在大多数生产系统中，DBA 很少甚至不能更改源代码，因而也就限制了他们调优应用程序的能力。这在 DBA 使用第三方工具时更是如此。所以，通常最有效的调优方法是解决问题的根源，即从 SQL 语句本身入手。通过查找哪些 SQL 语句消耗的资源最多来获得最佳性能，然后决定采取一定的措施来减少资源消耗。

通常，在第一次安装数据库时，会将其性能调至最优，但随着时间的流逝，一些常用的东西开始变得越来越慢。这在拥有大量数据的系统(例如决策支持系统)中尤为如此。用户开始发现诸如锁升级、全表扫描以及排序这样的因素造成性能下降，这些操作往往会迫使系统访问磁盘而不是访问内存。当出现这种情况时，最好检查一下 SQL，看看可以做哪些改进。

我们举这个事件监控器案例，旨在解决的问题是：针对通常的活动，调优正在用于最频繁访问数据库的 SQL，最消耗资源的 SQL。为了简化如何监控应用程序中 SQL 语句的问题，我们通过 DB2 的事件监视器，确定哪些 SQL 语句消耗的资源最多。

运行事件监视器

首先，必须创建事件监视器，然后运行监视器来收集将要分析的数据。

打开新的 DB2 命令行处理器会话，然后执行以下 DB2 命令：

```
db2 => update monitor switches using statement on
db2 => create event monitor sql_trace for statements write to table
```

创建完之后，我们可以看到，DB2 自动生成了下面这张表：

```
D:\>db2 list tables for all | find /i "stmt"
 STMT_SQL_TRACE                    ORACLE          T
 2008-10-09-15.09.35.781000
 db2 => set event monitor sql_trace state=1
```

(1) 创建事件监视器。

(2) 使该会话一直处于打开状态，直到这些数据库活动完成。确保 STMT_SQL_TRACE 所在的表空间有足够的存储空间，在此时间有可能会产生大量的数据。表空间的大小取决

于用户想要捕获的 SQL 语句的数目和交易量。

(3) 执行正常的数据库活动，直到您想监控的时段结束。这一监控阶段可以是问题产生时期，也可以是通常的数据库高峰期间。

(4) 回到在步骤(1)中打开的会话，然后发出以下语句：

```
db2 => set event monitor sql trace state=0
db2 => terminate
```

分析输出

由于已经在数据库表 STMT_SQL_TRACE 中存储了所需要的信息，因此可以查询该表以确定“讨厌”且耗时的 SQL 语句。

需要确定 4 类 SQL 语句。在执行以下查询以确定这些语句之前，在数据库配置参数中，至少要为应用程序堆大小(applheapsz)分配 256 个页面。

- 按照执行时间降序排列执行耗时最长的 SQL 语句。为了确定这些语句，使用下面的 SQL SELECT 语句：

```
select stmt text,(stop time-start time) "ExecutionTime(sec)"
from stmt sql trace
where stmt operation not in(7, 8, 9, 19 )
order by decimal(ExecutionTime) desc
fetch first 10 rows only
```

- 按照频率降序排列执行次数最多的 SQL 语句。可以用下面这条查询来确定这些语句：

```
select distinct(stmt text),count(*) Count from stmt sql trace
where operation not in(7,8,9,19)
group by stmt text
order by count(*)desc
fetch first 10 rows only
```

- 按照 CPU 时间降序排列最耗 CPU 时间的 SQL 语句。可以用下面这条查询来确定这些语句：

```
select stmt text ,user cpu time "UserCPU(sec)" from stmt sql trace
where operation not in(7,8,9,19)
order by usrcpu desc
fetch first 10 rows only
```

● 按照总排序时间降序排列排序时间最长的 SQL 语句。可以用下面这条查询找到这些语句：

```
select stmt text ,total sort time "TotalSortTime(ms)" from stmt sql trace
where operation not in(7,8,9,19 )
order by decimal(total sort time) desc
fetch first 10 rows only
```

捕获每一类中的 SQL 语句，并将它们放到 tune.sql 文件中。将下面这行插入到该文件中，这样可以更改工作负载中每条语句的执行频率：

```
--#SET FREQUENCY <x>
```

这里的<x>表示随后要执行 SQL 语句的次数。您的 tune.sql 文件类似于这样：

```
--#SET FREQUENCY 100
SELECT COUNT(*) FROM EMPLOYEE;
SELECT * FROM EMPLOYEE WHERE LASTNAME='HAAS';
--#SET FREQUENCY 1
SELECT AVG(BONUS), AVG(SALARY) FROM EMPLOYEE
GROUP BY WORKDEPT ORDER BY WORKDEPT;
```

将这些 SQL 语句复制到 tune.sql 中之后，检查任何 SQL 语句的 WHERE 子句中是否具有参数标志符(?)。将参数标志符改为适当的数据类型值，以便在没有任何错误的情形下执行 SQL 语句。

为了确定哪些索引可能提高性能，按如下脚本执行索引顾问程序：

```
$cd /sapr3/prod
$db2advis -d sample -i tune.sql -t 0 -o tuneidx.sql
```

所有推荐的索引将被放置在文件 tuneidx.sql 中。编辑该文件，在文件开始处添加一条连接语句：

```
connect to sample user userid using password;
```

在该文件的末尾添加下面这行：

```
terminate;
```

现在可以运行该文件以创建推荐的索引：

```
$db2 -tf tuneidx.sql -z tuneidx.log
```

注意：

其中，tuneidx.log 捕获 tuneidx.sql 的所有输出。

8.4.3 编写脚本从事件监控器中获取监控信息

如下脚本监控某段时间执行的 SQL 语句的执行性能信息，包括语句的执行时间、已经读取的行数和返回行数等信息。

```
#!/usr/bin/ksh
#shell script name is getlongsql.sh

if [ $# -eq 0 ] ; then
 echo "Usage:getlongsql.sh <database1> <interval>"
 exit
fi

DB=$1
INTERVAL=$2
FILENAME=sql.txt

connectdb()
{
db2 GET CONNECTION STATE|grep 'Connected'>/dev/null
res2=$?

if [[ $res2 -eq 0 ]]
then
  echo "The DB $DB has been connected!"
else
  db2 connect to $DB
  res=$?
  i=0
  while [ $i -lt 3 ]
  do
        i='expr $i + 1'
              if [[ $res -eq 0 ]]
              then
                echo "The DB $DB has been connected!"
                break
              else
                sleep 3
                db2 connect to $DB
                res=$?
```

```
                fi

      done
    fi
    }

    openEventMonitor()
    {
    db2 "drop table STMT SQLMON";
    db2 "drop table CONTROL SQLMON";
    db2 "drop table CONNHEADER SQLMON";
    db2 "drop event monitor SQLMON";
    db2 "create event monitor SQLMON for statements write to table";
    db2 "set event monitor SQLMON state 1"
    sleep $INTERVAL
    db2 "set event monitor SQLMON state 0"
    db2 "select START TIME, STOP TIME, (STOP TIME-START TIME) as
dur,ROWS READ,FETCH COUNT,substr(stmt text,1,400) as stmt from STMT SQLMON
order by dur desc" > $FILENAME
    more $FILENAME
    }

    connectdb
    openEventMonitor
```

执行结果：

```
    START TIME                  STOP TIME                  DUR
ROWS READ           FETCH COUNT         STMT


-------------------------- ----------------------------------------------------
    2013-01-30-10.57.13.871797 2013-01-30-10.57.13.909544
0.037747                 0                 17 SELECT TBSP TYPE, TBSP NAME FROM
SYSIB MADM.TBSP UTILIZATION where DBPARTITIONNUM = 0



    2013-01-30-10.57.13.871797 2013-01-30-10.57.13.908193
0.036396                 0                 17 SELECT TBSP TYPE, TBSP NAME FROM
SYSIB MADM.TBSP UTILIZATION where DBPARTITIONNUM = 0
    2013-01-30-10.57.13.909598 2013-01-30-10.57.13.910341
0.000743                 0                  0
    2013-01-30-10.57.11.412991 2013-01-30-10.57.11.413198
0.000207                 0                  1 SELECT COUNT(*) FROM SYSIBM.dual
    2013-01-30-10.57.11.414702 2013-01-30-10.57.11.414865
```

```
0.000163                  0                  1 SELECT COUNT(*) FROM SYSIBM.dual
    2013-01-30-10.57.11.435734 2013-01-30-10.57.11.435884
0.000150                  0                  1 SELECT COUNT(*) FROM SYSIBM.dual
    2013-01-30-10.57.13.871577 2013-01-30-10.57.13.871707
0.000130                  0                  0 SELECT TBSP_TYPE, TBSP_NAME FROM
SYSIB
    MADM.TBSP_UTILIZATION where DBPARTITIONNUM = 0
```

8.5 db2mtrk 及监控案例

db2mtrk 是用于在 DB2 数据库中进行内存跟踪的工具，可以用于查看实例、数据库、代理进程当前对内存的使用状态。

例 8-1 db2mtrk 监控示例 1。

```
    db2mtrk -i -d -v
    Memory for database:  SAMPLE
    Backup/Restore/Util Heap is of size 16384 bytes
    Package Cache is of size 81920 bytes
    Catalog Cache Heap is of size 65536 bytes
      Buffer Pool Heap is of size 4341760 bytes
      Buffer Pool Heap is of size 655360 bytes
      Buffer Pool Heap is of size 393216 bytes
      Buffer Pool Heap is of size 262144 bytes
      Buffer Pool Heap is of size 196608 bytes
      Lock Manager Heap is of size 491520 bytes
      Database Heap is of size 3637248 bytes
      Other Memory is of size 16384 bytes
      Application Control Heap is of size 327680 bytes
      Application Group Shared Heap is of size 57344000 bytes
    Total: 67829760 bytes
```

例 8-2 db2mtrk 监控示例 2。

```
    C:\>db2mtrk -i -d -p -r 1200
    在 2012/10/15 10:10:49 跟踪内存
    用于实例的内存
     other       fcmbp       monh
     13.3M       768.0K      320.0K
    用于数据库 SAMPLE 的内存
     utilh       pckcacheh   other       catcacheh   bph(1)      bph(S32K)
     64.0K       384.0K      128.0K      128.0K      2.3M        832.0K
     bph(S16K)   bph(S8K)    bph(S4K)    shsorth     lockh       dbh
```

```
 576.0K      448.0K      384.0K      0           320.0K      12.2M
apph(15)    apph(14)    apph(13)    apph(12)    apph(11)    appshrh
64.0K       64.0K       64.0K       64.0K       64.0K        128.0K
用于代理程序 3276 的内存
other
      192.0K
用于代理程序 2964 的内存
   other
   192.0K
用于代理程序 4128 的内存
   other
    576.0K
用于代理程序 4332 的内存
   other
    192.0K
```

db2mtrk 工具的语法如下：

```
>>-db2mtrk--+-----+--+-----+--+-----+--+-----+--+-----+--------->
           '- -i-'  '- -d-'  '- -p-'  +- -m-+  '- -a-'
                                       '- -w-'

>--+-----------------------+--+-----+--+-----+-------------><
   '- -r--interval--+-------+-'  '- -v-'  '- -h-'
                   '-count-'
```

- db2mtrk -i　　　#显示当前实例的内存使用情况。
- db2mtrk -i -v　#显示当前实例的内存使用的详细信息。
- db2mtrk -d　　　#显示数据库的内存使用情况。
- db2mtrk -d -v　#显示数据库的内存使用情况的详细信息。
- db2mtrk -p　　　#显示代理进程的专用内存使用率。
- db2mtrk -h　　　#显示帮助信息。

-m 参数选项用于显示最大的内存使用上线。

-w 参数选项用于显示使用过程中内存达到的最大值，即 watermark。

-r 参数选项用于重复显示，其中 interval 是重复显示的时间间隔数，count 是要重复显示的次数。

interval 指定以秒为单位的重复间隔。

count 指定间隔的次数。

-v 详细输出。

-h 显示帮助信息。

注意：

在我们使用 db2mtrk 工具时，主要查看高水位和实际的配置之间是否接近。例如，如果 util_heap_sz 的高水位逼近实际的为数据库配置参数指定的值，那说明您可以考虑增大该参数。

8.6 本章小结

在 DB2 日常运维过程中，数据库的性能需要我们定期监控并收集信息，然后根据这些信息对数据库进行调整，以便达到较好的性能。所以监控数据库对运维至关重要，合理选择适合的监控工具，将帮助我们快捷获取有用的信息，希望本章的案例能启发各位在日常运维工作中熟练和正确运用这些监控工具。

第 9 章

DB2 故障诊断

首先大家要把“诊断”和“性能”这两个术语区分开(当然有的时候它们往往是有关联的)，故障诊断是数据库中最艰难的工作之一。例如，你的应用程序异常挂起了，你的应用程序时断时续等。如何定位问题所在呢？在复杂的系统中，应用程序、中间件、DB2 和操作系统每个环节都可能有问题。诊断会有很大的难度，本章主要讲解在 DB2 层面如何来进行数据库诊断。

本章主要讲解如下内容：

- DB2 故障诊断机制
- 故障诊断文件
- 故障诊断工具
- 故障诊断流程

9.1　DB2 故障诊断机制

9.1.1　故障诊断相关文件

FODC 表示首次故障数据捕获，是 DB2 内部的故障处理机制。FODC 保留处理故障生成的信息，并将控制返回到受影响的引擎。捕获的数据保存在日志文件中，供问题分析之用。FODC 主要供 IBM Service 和 DBA 查看，不仅仅应用在 DB2 产品中，也同样应用于

WebSphere Application Server、MQ 和 AIX 等 IBM 相关产品中。运行时出现的事件和错误，信息一出现即加以捕获，并将其写入日志文件中，供 DBA 进行分析。FODC 架构如图 9-1 所示：

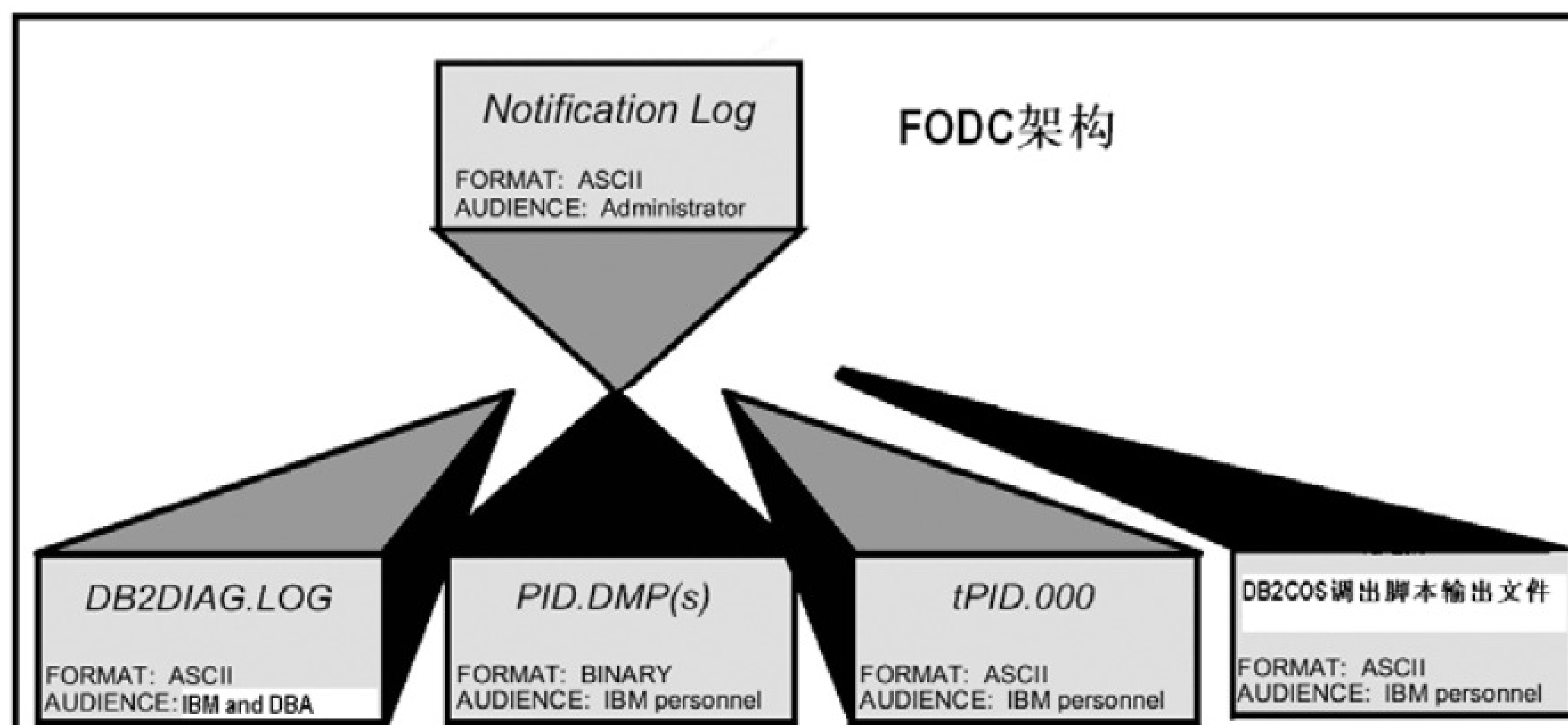


图 9-1 FODC 架构图

首次出现数据捕获(FODC)是用来捕获有关 DB2 数据库的基于场景的数据的过程。DB2 用户可根据特定症状手动调用 FODC，也可以通过配置要捕获的事件类型(例如数据库挂起)，在事件发生时自动调用 FODC 去捕获当时的各项数据信息，以便事后进行问题分析。不然可能需要手动重现问题，然后收集信息，再去分析。

在诊断路径 DIAGPATH 下找到有关问题诊断的文件，这些文件中存放有关 FODC 的信息。诊断路径可以通过下面的命令获取，也可更改 DBM 配置以修改默认路径。

```
$db2 GET DBM CFG|grep DIAG
 Diagnostic error capture level  (DIAGLEVEL) = 3
 Diagnostic data directory path  (DIAGPATH) = /home/db2inst1/sqllib/db2dump/
```

和诊断有关的 FODC 日志文件主要有以下几个：

1. 管理通知日志(Notification Log)

发生重大事件时，DB2 将信息写入管理通知(“instance_name.nfy”)日志。该信息供数据库和系统管理员使用。通知消息提供了其他信息以补充提供的 SQLCODE。事件的类型和收集信息的级别控制是由 NOTIFYLEVEL 配置参数确定的。

DB2 数据库管理器将下列类型的信息写入管理通知日志：DB2 实用程序(如 REORG 和 BACKUP)的状态；客户机应用程序错误、服务类更改、许可证发放活动、日志文件路径和存储问题、监视活动并为活动建立索引，以及表空间问题。数据库管理员可以使用这

些信息来诊断问题、调整数据库或仅仅监视数据库。

管理通知日志消息还以标准化的消息格式记录到 db2diag.log。此日志只在 Linux/UNIX 平台上存在，在 Windows 平台上不存在。

2. DB2 诊断日志(“db2diag.log”)

此文本文件包含关于实例遇到的错误和警告的诊断信息。此信息用于故障诊断，且旨在供 DBA 使用。记录在此文件中的消息类型由 *DIAGLEVEL* 数据库管理器配置参数确定。

3. 转储文件(DUMP 文件)

对于某些错误情况，会将附加信息记录在以失败进程标识命名的二进制文件中。这些文件主要供 IBM 软件支持机构使用。原始堆栈转储可能包括在 ASCII 陷阱文件中。特定于数据库管理器内的组件的转储文件存储在相应 FODC 包目录中。

转储文件是在发生错误时创建的，包含将有助于诊断问题(例如内部控制块)的其他信息。写至转储文件的每个数据项都具有与其相关联的时间戳记，以帮助进行问题确定。转储文件使用二进制格式，目的是供 IBM 内部人员进一步定位是 bug 还是程序代码错误等。这个文件需要用到 IBM 内部工具，所以一般我们是不能读的。

创建或追加转储文件时，会在 db2diag.log 中建立条目，指示所写数据的时间和类型。这些 db2diag.log 条目类似于以下所示：

```
2012-09-11-18.49.09.155510+480 I44101A410        LEVEL: Error
PID     : 23658614           TID  : 1          PROC : YQrzqs
INSTANCE: db2yst              NODE : 000
EDUID   : 1
FUNCTION: DB2 UDB, trace services, sqlt logerr data (secondary logging func,
probe:0
MESSAGE : dbm config
DATA #1 : Dumped object of size 16384 bytes at offset 925120, 51 bytes
/home/db2yst/sqllib/db2dump/23658614.1.000.dump.bin
```

其实，这个文件对 DBA 而言基本上是不可读的，这个文件主要是在数据库出现重大问题时，我们把这个文件收集并发送给 IBM 技术支持机构以判断是应用程序问题还是数据库 bug。这个问题必须使用 IBM 内部工具才能读，所以说虽然这个文件比较重要，不过通常我们很少用到。

4. 陷阱(trap)文件

如果 DB2 由于陷阱、分段违例或异常而不能继续处理，就会生成陷阱文件。DB2 接

收到的所有信号或异常都会记录在陷阱文件中。陷阱文件还包含发生错误时正在运行的函数序列，此序列有时又称“函数调用堆栈”或“堆栈跟踪”。陷阱文件还包含有关捕获到信号或异常时进程状态的其他信息。文件位于由 DIAGPATH 数据库管理器配置参数指定的目录中。陷阱文件包含如下信息：

- 可用虚拟内存量
- 发生陷阱时与产品的配置参数及注册变量相关的值
- 发生陷阱时 DB2 产品使用的估计内存量
- 提供中断上下文的信息

在所有平台上，陷阱文件名以进程标识(PID)开头，后跟线程标识(TID)，再后跟分区号(在单分区数据库上为 000)，并以“.trap.txt”结尾。

还有一些诊断陷阱，它们是在发生某些特定条件(这些条件不一定会使实例崩溃)时由代码生成的，在查看堆栈时非常有用。这些陷阱是使用 PID 以十进制格式命名的，后跟分区号(在单分区数据库中为 0)。

例如：

- 6881492.2.000.trap.txt 是进程标识(PID)为 6881492、线程标识(TID)为 2 的陷阱文件。
- 6881492.2.010.trap.txt 是进程和线程在分区 10 上运行的陷阱文件。

可将 db2pd(这个命令在后面会讲解)命令与-stack all 或-dump 选项配合使用，以根据需要生成陷阱文件。目的是供 IBM 内部人员来进一步定位是 bug 还是程序代码错误等等。其实，这个文件和转储文件一样，对 DBA 而言基本上是不可读的，这个文件主要是在数据库出现重大问题时，我们把这个文件收集并发送给 IBM 技术支持机构以判断是应用程序问题还是数据库 bug。这个问题必须使用 IBM 内部工具才能读，所以说虽然这个文件比较重要，不过通常我们很少用到。

5. DB2 调出脚本(db2cos)输出文件

db2cos 脚本提供给我们用于高级复杂诊断的脚本，db2cos 脚本和 db2pd 结合起来可以为 DBA 提供强大的故障诊断工具。在默认情况下，当数据库管理器因为应急启动、陷阱、分段违例或异常而不能继续处理时，将会自动调用 db2cos 脚本。每个默认 db2cos 脚本将调用 db2pd(关于这个强大的工具，在本章后面会讲解)命令以打开方式收集信息。db2cos 脚本名称的格式为 db2cos_hang、db2cos_trap 等等。每个脚本的行为方式类似，但 db2cos_hang 有所不同，它是通过 db2fodc 工具调用的。默认 db2cos 脚本将调用 db2pd 命令以打开方式收集信息。根据 db2cos 脚本中包含的命令，db2cos 输出文件的内容会有所不同。

默认 db2cos 脚本在 bin 目录中。在 UNIX 操作系统中，此目录是只读的。可将 db2cos

脚本复制到 adm 目录，必要时可在此位置修改该文件。如果 db2cos 脚本在 adm 目录中，就运行该脚本，否则会运行 bin 目录中的脚本。

6. AIX 平台上默认的 db2cos 脚本的内容

```
#!/bin/sh

# This is a data collection script for generic troubleshooting that can
# be invoked by the engine and also configured to be invoked on demand
# (see db2pdcfg -catch and other tools to invoke it).

# In some cases it might be needed to modify this script to collect extra
# information (such as traces) or execute additional actions.
# To do so place a copy of this file in sqllib/adm and modify it there.
# The script in sqllib/bin will be executed only when the customized script
# is not found in sqllib/adm.
#
# This script generates a logfile with extension .cos.txt in $diagpath
#

# Save arguments to write them in the logfile
arguments="";

###########################################################################
###
###############################   PARSING PART
############################
###########################################################################
###

# In this parsing we have to recognize all the arguments. Do not want to allow
# parameters without '=' sign, or without being correctly parsed
while [ "$#" -gt "0" ]
do
   TAG=; VALUE=;
   TAG=`echo $1| sed 's/=.*//'`; VALUE=`echo $1| sed 's/'$TAG'=*//'`;
   arguments="$arguments\n $1";

   if [ ! -n "$VALUE" ]
   then
      arguments="$arguments-- Bad formed string"
   else
      case $TAG in
```

```
    # Here we recognize the parameters that are common to all automatic callout
    # scripts. These are passed by the engine
      INSTANCE)       instance=$VALUE       ;;
      DATABASE)       database=$VALUE       ;;
      DBPART)         dbpart=$VALUE         ;;
      PID)            pid=$VALUE            ;;
      TID)            tid=$VALUE            ;;
      EDUID)          eduid=$VALUE          ;;
      FUNCTION)       function=$VALUE       ;;
      COMPONENT)      component=$VALUE      ;;
      PROBE)          probe=$VALUE          ;;
      TIMESTAMP)      timestamp=$VALUE      ;;
      APPID)          appid=$VALUE          ;;
      APPHDL)         apphdl=$VALUE         ;;
      DIAGPATH)       diagpath=$VALUE       ;;
      REASON)         reason=$VALUE         ;;
      DESCRIPTION)    description=$VALUE    ;;
      DB2 DEBUG)      db2 debug=$VALUE      ;;
      OS)             OS=$VALUE             ;;

      *) # If it is unrecognized add a comment to the log file
        arguments="$arguments Unrecognized TAG: $TAG"
        ;;
    esac
  fi
  shift
done

##########################################################################
###
#############################   PREPARATION PART
##########################
##########################################################################
###

# Prefix for files. Since this script can be executed several times, we want
# to have a unique prefix for each execution. All files created should use it.
prefix="$pid.$eduid.$dbpart"

# Output file must not change as the engine will also be writing to this same
# file.  Changing this will cause db2cos script and engine to be out of sync.
logfile=$diagpath/$prefix.cos.txt;
```

```
# Print header of the log file
echo '<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="db2fodc.xsl" type="text/xsl"?>
<db2fodc>
<Header>
<type>Generic db2cos</type>
</Header>
<Parameters>'                    >> $logfile;
echo $arguments                  >> $logfile;
echo "</Parameters>"              >> $logfile;
echo "Started at: `date +%X`"    >> $logfile;
# Save the initial time in order to know how long this script takes
initial time=`date +%s`

##########################################################################
###
############################   EXECUTION PART
#############################
##########################################################################
###

# Hand control over to the rest of the script
# $1 is the first "other argument"
case "$reason" in
  "DATA COR")
    # This will never get executed. Check db2cos datacorruption script
  ;;
  "TRAP")
    echo "Trap Caught"                                       >> $logfile
    if [ ! -n "$database" ]
    then
      db2pd -inst                                          >> $logfile
    else
      db2pd -db $database -inst                              >> $logfile
    fi
  ;;
  "DUMP")
    echo "Dump Caught"                                       >> $logfile
    if [ ! -n "$database" ]
    then
      db2pd -inst                                          >> $logfile
    else
      db2pd -db $database -inst                              >> $logfile
```

```
    fi
  ;;
  "DB2 TRC")
    echo "Trace Point Caught"                            >> $logfile
    # trace area
    # Script was invoked from inside a trace point

    # Uncomment this if you'd like trace to be turned off
    # db2trc dump /tmp/trc.dmp
    # db2trc off
  ;;
  "LOCKTIMEOUT")
    echo "Lock Timeout Caught"                           >> $logfile
    if [ ! -n "$database" ]
    then
      db2pd -inst                                       >> $logfile
    else
      db2pd -db $database -inst                          >> $logfile
    fi
  ;;
  "DEADLOCK")
    echo "Lock Deadlock Caught"                          >> $logfile
    if [ ! -n "$database" ]
    then
      db2pd -inst                                       >> $logfile
    else
      db2pd -db $database -inst                          >> $logfile
    fi
  ;;
  "SQLCODE")
    echo "Sqlcode Caught"                               >> $logfile
    if [ ! -n "$database" ]
    then
      db2pd -inst                                       >> $logfile
    else
      db2pd -db $database -inst                          >> $logfile
    fi
  ;;
  "ADMCODE")
    echo "ADM Code Caught"                              >> $outfile
    if [ ! -n "$database" ]
    then
      db2pd -inst                                       >> $outfile
```

```
    else
      db2pd -db $database -inst                              >> $outfile
    fi
  ;;
  "DIAGSTRING")
    echo "Diag String Caught"                                >> $outfile
    if [ ! -n "$database" ]
    then
      db2pd -inst                                          >> $outfile
    else
      db2pd -db $database -inst                              >> $outfile
    fi
  ;;
  *)
    echo "Unknown type!"                                    >> $logfile;
  exit
  ;;
esac

###########################################################################
###
############################   FINISH PART
################################
###########################################################################
###

# Report the time elapsed since we started the script and write we finished
end time='date +%s'
elapsed time='expr $end time - $initial time'
echo "Elapsed time in the script: $elapsed time seconds" >> $logfile;
echo '</db2fodc>'                                  >> $logfile;
```

通过上面的脚本可以看到，对于数据库级别的事件或故障，默认的 db2cos 脚本用-db 和-inst 选项调用 db2pd。DBA 用 db2pd 调用替换相应的行，调用收集锁超时分析所需的监视器数据。

还可通过 db2pdcfg -cos 命令来配置触发 db2cos 调用的信号类型。默认配置用于在发生应急启动或陷阱时运行的 db2cos 脚本。但是在默认情况下，生成的信号不会启动 db2cos 脚本。例如，下面是使用 db2pdcfg 收集锁超时的配置。

为了每当出现锁超时时启动 db2cos 脚本，DBA 调用 db2pdcfg 实用程序，如下所示：

```
    db2pdcfg -catch locktimeout count=1
```

-catch 选项指定应该自动调用 db2cos 脚本的故障或事件。对于锁超时事件，可以指定字符串 locktimeout，也可以指定与锁超时相应的 SQL 错误码和原因码：

```
db2pdcfg -catch 911,68 count=1
```

发生应急启动、陷阱、分段违例或异常时，事件顺序如下：

(1) 创建陷阱文件

(2) 调用信号处理程序

(3) 调用 db2cos 脚本(取决于启用的 db2cos 设置)

(4) 在管理通知日志中记录相应条目

(5) 在 db2diag.log 中记录相应条目

db2cos 脚本中的 db2pd 命令收集到的默认信息包括有关操作系统、已安装 DB2 产品的版本和服务级别、数据库管理器和数据库配置的详细信息，以及有关以下各项的状态的信息：代理程序、内存池、内存块、应用程序、实用程序、事务、缓冲池、锁定、事务日志、表空间和容器。此外还会提供有关下列各项的信息：动态、静态和目录高速缓存的状态、表和索引统计信息、恢复状态以及重新优化的 SQL 语句及活动语句列表。如果需要收集进一步的信息，那么只需要使用附加命令更新 db2cos 脚本即可。

调用默认 db2cos 脚本时，将在 DIAGPATH 数据库管理器配置参数指定的目录中生成输出文件。这些文件名为 XXX.YYY.ZZZ.cos.txt，其中 XXX 是进程标识(PID)，YYY 是线程标识(TID)，而 ZZZ 是数据库分区号(对于单分区数据库则为 000)。如果存在多线程陷阱，那么会对每个线程单独调用 db2cos 脚本。如果 PID 和 TID 组合多次出现，那么数据将追加至文件。还会显示时间戳记，所以你可以区分输出的迭代。

9.1.2 设置故障诊断级别

1. 设置管理通知日志文件的错误捕获级别

在 UNIX 平台上，管理通知日志是名为 instance.nfy 的文本文件。在 Windows 平台上，所有管理通知消息都写到“事件日志”中。错误可由 DB2、“运行状况监视器”、Capture 和 Apply 程序以及用户应用程序写入。DB2 记录在管理通知日志中的信息由 NOTIFYLEVEL 设置确定。

- 要检查当前设置，发出 db2 get dbm cfg|grep NOTIFYLEVEL 命令。

```
Notify Level (NOTIFYLEVEL) = 3
```

- 要改变该设置，使用 UPDATE DBM CFG 命令。 例如：

```
db2 UPDATE DBM CFG USING NOTIFYLEVEL X
```

其中，X 是期望的通知级别。X 的有效值为：

- **0**：未捕获任何管理通知消息(不建议使用此设置)。
- **1**：致命或不可恢复错误。仅记录致命和不可恢复错误。要从这些情况中的某些情况进行恢复，可能需要来自 DB2 服务机构的帮助。
- **2**：需要立即操作。记录了需要系统管理员或数据库管理员立即注意的情况。如果未注意该情况，那么可能会导致致命错误。非常重要并且没有错误的活动(例如恢复)的通知可能也在此级别记录。此级别将捕获“运行状况监视器”警报。
- **3**：重要信息，不需要立即操作。记录没有威胁也不需要立即操作，但是可能表示并非最佳系统的情况。此级别将捕获“运行状况监视器”警报、“运行状况监视器”警告和“运行状况监视器”引起的注意信息。
- **4**：参考消息。

将通知级别设置为 3 将导致管理通知日志包括可应用于级别 1、2 和 3 的消息。

建议：你可能希望增大通知级别的值以收集更多问题确定数据来帮助解决问题。注意，必须将通知级别的值设置为 2 或更高，以便“运行状况监视器”向在其配置中定义的联系人发送所有通知。

备注：通知级别的级别调整是可联机配置的。

2. 设置诊断日志文件的错误捕获级别

DB2 诊断日志是包含 DB2 记录的文本信息的文件。此信息可用于故障诊断，但主要用于 DB2 客户支持。DB2 在 db2diag.log 中记录的信息由 DIAGLEVEL 设置确定。

- 要检查当前设置，发出 db2 get dbm cfg|grep DIAGLEVEL 命令：

```
Diagnostic error capture level (DIAGLEVEL) = 3
```

- 要联机更改数据库管理器配置参数，请使用以下命令：

```
db2 update dbm cfg using <parameter-name>  <value>
```

例如：

```
db2  UPDATE DBM CFG USING DIAGLEVEL X
```

其中，X 是期望的诊断级别。

如果要诊断可再现的问题，支持人员建议在执行故障诊断时使用 DIAGLEVEL 4，此参数的有效值为：

- **0**：没有捕获到诊断数据
- **1**：仅严重错误

- **2**：所有错误
- **3**：所有错误和警告
- **4**：所有错误、警告以及参考消息

建议：在应用上线初期为了仔细定位问题，可以考虑把诊断级别设置为 4 以收集详细信息，等应用运行稳定后再把诊断级别调低。

备注：诊断级别的级别调整是联机配置的。

3. 收集故障诊断信息

发生影响实例的中断时，可自动将诊断信息收集到包中，包中的信息也可手动创建。

4. 自动收集诊断信息

在出现故障时，数据库管理器调用 db2fodc 命令以自动执行首次出现数据捕获(FODC)。

为了使中断与 DB2 诊断日志和其他故障诊断文件相关，会同时将诊断消息写至管理通知日志和 db2diag.log。诊断消息包括 FODC 目录名和创建 FODC 目录时的时间戳记。FODC 包描述文件在新的 FODC 目录中。

5. 配置诊断信息的自动收集

必须先指示数据库管理器要采取的操作，数据库管理器才能自动执行操作。系统会设置一些标志，指示数据库操作期间遇到错误或警告时数据库管理器应采取的操作。要采取的操作包括：

- 在 db2diag.log 中生成堆栈跟踪(默认)。
- 运行标注脚本 db2cos(默认)。
- 停止跟踪(db2trc)命令。

6. 更改首次出现数据捕获(FODC)选项

使用“配置 DB2 数据库的问题确定行为”(db2pdcfg)命令来更改首次出现数据捕获(FODC)选项。FODC 选项是使用 db2pdcfg 工具在 DB2FODC 注册变量中设置的。这些选项会影响 FODC 情况中有关数据捕获的数据库系统行为。

7. 诊断信息的手动收集

“首次出现数据收集”(db2fodc)命令用于收集有关可能中止的信息或出现严重性能问题时的信息。运行 db2fodc 命令后，会在当前诊断路径下自动创建新目录 FODC_hang_<timestamp>，还会运行 db2cos_hang 脚本。此脚本控制将收集并放在 FODC 子目录中的数

据集合。FODC 子目录是否存在取决于 db2fodc 命令的运行方式或 DB2 注册变量的配置。

那么，什么时候需要手工收集诊断信息呢？

db2fodc 手工运行，可以用来收集性能、挂起和索引错误 3 类问题：

```
Command syntax
>>-db2fodc--+-+- -hang-+--+-----------------+--------+--------->
            | '- -perf-'  |       .-,------. |        |
            |             |       V        | |        |
            |             +- -db----dbname-+-+        |
            |             '- -alldbs---------'        |
            '- -indexerror--FODC IndexError directory-'
>--+-------+--+---------------------------+--------------------->
   +-basic-+  '- -fodcpath--fodc path name-'
   '-full--'
```

示例：

```
/db2/db2ara/sqllib/db2dump$db2fodc -perf full -db db2ara
*********************WARNING**********************
*    This tool should be run with caution.       *
*    It can cause some performance degradation,   *
*     especially on busy systems with a          *
*       high number of active connections.       *
**************************************************

You have 10 seconds to cancel this script with Ctrl-C

Executing preparation for performance collection
--------------------------------------------------------------
 >> Output directory:
/db2/db2ara/sqllib/db2dump/FODC Perf 2012-09-03-17.35.14.222119 0000

 >> This execution will monitor the system for approximately 17 minutes and
10 seconds
 >> Summary of the execution:
  Tool to run      Number of samples
     run vmstat        200
     run iostat        200
     run snapshot       6
     run stacktrace     5
     run db2perfcount   2
     run db2trc         1
     run_dpsdbcb        1
```

```
    Data Collection Start:  Mon Sep 3 17:35:29 GMT+08:00 2012
    --------------------------------------------------------------
    at 0 seconds: snapshot - Executing
    at 1 seconds: vmstat - Executing
    at 1 seconds: iostat - Executing
    at 2 seconds: snapshot - Finished

    at 73 seconds: stacktrace - Executing
    at 119 seconds: stacktrace - Finished
    at 146 seconds: snapshot - Executing
    at 146 seconds: snapshot - Finished
    at 219 seconds: stacktrace - Executing
    at 265 seconds: stacktrace - Finished
    at 292 seconds: db2perfcount - Executing
    at 303 seconds: db2perfcount - Finished
    at 365 seconds: snapshot - Executing
    at 366 seconds: snapshot - Finished
    at 438 seconds: stacktrace - Executing
    at 484 seconds: stacktrace - Finished
    at 511 seconds: snapshot - Executing
    at 511 seconds: snapshot - Finished
    at 584 seconds: stacktrace - Executing
    at 630 seconds: stacktrace - Finished
    at 657 seconds: snapshot - Executing
    at 658 seconds: snapshot - Finished
    at 730 seconds: db2trc - Executing
    at 741 seconds: db2trc - Finished
    at 803 seconds: dpsdbcb - Executing
    at 803 seconds: dpsdbcb - Finished
    at 876 seconds: db2perfcount - Executing
    at 887 seconds: db2perfcount - Finished
    at 949 seconds: stacktrace - Executing
    at 995 seconds: stacktrace - Finished
    at 1001 seconds: iostat - Finished
    at 1001 seconds: vmstat - Finished
    at 1022 seconds: snapshot - Executing
    Finished all snapshots
    at 1022 seconds: snapshot - Finished

    /db2/db2ara/sqllib/bin/db2cos perf Finished. Exiting at Mon Sep  3 17:53:44
GMT+08:00 2012...
```

```
    Output directory is
/db2/db2ara/sqllib/db2dump/FODC Perf 2012-09-03-17.35.14.222119 0000
    Open db2fodc.log in that directory for details of collected data
```

8. FODC 数据及其放置

DIAGPATH 配置参数用来指定目录，其中将包含根据 diaglevel 参数的值可能会生成的错误文件、事件日志文件、警报日志文件以及任何转储文件。

根据平台，此目录可能包含转储文件、陷阱文件、错误日志、通知文件、警报日志文件和“首次出现数据集合”(FODC)程序包。

如果此参数为 Null，那么诊断信息将写入下列其中一个目录或文件夹的文件中：

- 在 Windows 环境中： 用户数据文件(例如实例目录下的文件)被写入不同于安装了代码的位置，比如用户数据文件将被写入 C:\Documents and Settings\All Users\Application Data\IBM\DB2\Copy Name，其中的 Copy Name 是 DB2 副本的名称。
- 在 Linux 和 UNIX 环境中：信息将写入 *INSTHOME*/sqllib/db2dump，其中的 *INSTHOME* 是实例的主目录。

根据实例内的中断类型，“首次出现数据捕获”(FODC)会导致创建子目录及收集的特定内容，将创建一系列子目录及文件和日志的集合。

注意：
应定期清理转储目录以防止它变得太大。

9.2　深入讲解故障诊断文件

9.2.1　解释管理通知日志文件条目

使用文本编辑器来查看怀疑发生了问题的机器上的管理通知日志文件。记录的最新事件在文件的最后面。通常，每个条目包含下列部分：

- 时间戳记
- 报告错误的位置。应用程序标识允许在服务器和客户机的日志上找到与应用相关的条目
- 说明错误的诊断消息(通常以“DIA”或“ADM”开头)
- 任何可用的支持数据(例如 SQLCA 数据结构)和指向任何其他转储文件或陷阱文件的位置的指针

管理日志与数据库中的所有日志一样会不断增长。根据每个文件中记录的内容不同，某些日志的增长速度会比其他日志的增长速度快。日志过大时，应对其进行备份并清除。下一次系统需要新日志时会自动生成。

以下示例显示样本日志条目的标题信息，并且标识了日志的所有部分。

注意：
不是所有日志条目都包含所有这些部分。

```
2012-09-12-19.33.37.630000 1  实例：DB2 2  节点：000 3
PID：940(db2syscs.exe) TID：660 4  Appid：*LOCAL.DB2.020205091435 5
恢复管理器 6  sqlpresr 7  探测点：1 8  数据库：SAMPLE 9
ADM1530E 10  已启动崩溃恢复。 11
```

图注：

1 消息的时间戳记。

2 生成消息的实例的名称。

3 对于多分区系统，此项为生成消息的数据库分区。在非分区数据库中，值为“000”。

4 进程标识(PID)，后跟进程名称，再后跟负责生成消息的线程标识(TID)。

5 进程正在为其工作的应用程序的标识。在本例中，生成消息的进程代表标识为 *LOCAL.DB2.020205091435 的应用程序工作。

要标识关于特定应用程序标识的信息，执行下列其中一项操作：

- 在 DB2 服务器上使用 LIST APPLICATIONS 命令查看应用程序标识列表。可以根据此列表确定有关遇到错误的客户机的信息，例如节点名及其 TCP/IP 地址。
- 使用 GET SNAPSHOT FOR APPLICATION 命令查看应用程序标识列表。

6 写入消息的 DB2 组件。对于由使用 db2AdminMsgWrite API 的用户应用程序编写的消息，该组件将读取“用户应用程序”。

7 提供消息的函数的名称。此函数在写入消息的 DB2 组件中运行。对于由使用 db2AdminMsgWrite API 的用户应用程序编写的消息，该函数将读取“用户函数”。

8 唯一内部标识。此数字允许 DB2 客户支持和开发在报告了消息的 DB2 源代码中找到相应位置。

9 发生错误的数据库。

10 以十六进制代码指示错误类型和编号的消息(如果存在)。

11 说明记录的事件的消息文本(如果存在)。

9.2.2　解释诊断日志文件条目

使用 db2diag.log 分析工具(db2diag)过滤并格式化 db2diag.log 文件。虽然已使用标准化消息格式将管理通知日志消息记录至 db2diag.log，但还是建议先查看 db2diag.log 以了解数据库所发生的情况。

除了使用 db2diag 之外，还可使用文本编辑器来查看怀疑发生了问题的机器上的诊断日志文件。记录的最新事件在文件的最后面。

注意：

管理和诊断日志会不断增加。当里面的条目过多时，最好在将它们备份后再清除，系统会在写它们的时候自动重建。

以下示例显示样本日志条目的标题信息，并且标识了日志的所有部分。

注意：

不是所有日志条目都包括所有这些部分。只有开头的一些字段(时间戳记至 TID)和函数才会显示在所有 db2diag.log 记录中。

```
2016-09-11-14.20.46.973000-240 1  I27204F655 2  LEVEL: Info 3
PID : 3228 4  TID : 8796 5  PROC : db2syscs.exe 6
INSTANCE: DB2MPP 7  NODE : 002 8  DB : WIN3DB1 9
APPHDL : 0-51 10  APPID: 9.26.54.62.45837.070518182042 11
AUTHID : UDBADM 12
EDUID : 8796 13  EDUNAME: db2agntp 14  (WIN3DB1) 2
FUNCTION: 15  DB2 UDB, data management, sqldInitDBCB, 16  probe:4820
DATA #1 : 17  String, 26 bytes
Setting ADC Threshold to:
DATA #2 : unsigned integer, 8 bytes
1048576
```

图注：

1　消息的时间戳记和时区。

2　记录标识字段。对于创建 DB2 诊断日志的平台，db2diag.log 的记录标识指定要记录的当前消息的文件位移(如“27204”)和消息长度(如“655”)。

3　与错误消息相关联的诊断级别，例如参考、警告、错误、严重或事件。

4　进程标识。

5　线程标识。

6　进程名称。

7 生成该消息的实例的名称。

8 对于多分区系统，此项为生成该消息的数据库分区。在非分区数据库中，该值为“000”。

9 数据库名称。

10 应用程序句柄。此值与 db2pd 输出和锁定转储文件中使用的值相对应，包括后跟协调程序索引编号并且用破折号分开的协调程序分区号。

11 进程正在为其工作的应用程序的标识。在本例中，生成消息的进程代表标识为 9.26.54.62.45837.070518182042 的应用程序工作。

TCP/IP 生成的应用程序标识由 3 部分组成：

- IP 地址：表示为 32 位数字，最大显示为 8 位十六进制字符。
- 端口号：显示为 4 位十六进制字符。
- 此应用程序的实例的唯一标识。

注意：

当 IP 地址或端口号的十六进制版本以 0 至 9 开头时，它们将分别转换为 G 至 P。例如，“0”映射为“G”，“1”映射为“H”，依此类推。IP 地址 AC10150C.NA04.006D07064947 表示为如下所示：IP 地址仍为 AC10150C，它将转换为 172.16.21.12；端口号为 NA04。第一个字符为“N”，它将映射为“7”。因此，端口号的十六进制为 7A04，它将转换为十进制格式 31236。

要标识关于特定应用程序标识的信息，执行下列其中一项操作：

- 在 DB2 服务器上使用 LIST APPLICATIONS 命令查看应用程序标识列表。可以根据此列表确定有关遇到错误的客户机的信息，例如数据库分区名及其 TCP/IP 地址。
- 使用 GET SNAPSHOT FOR APPLICATION 命令查看应用程序标识列表。
- 使用 db2pd -applications -db <sample>命令。

12 授权标识。

13 引擎可调度单元标识。

14 引擎可调度单元的名称。

15 产品名(“DB2”)、组件名(“数据管理”)、写入消息的函数名(“sqlInitDBCB”)以及函数内的探测点(“4820”)。

16 DB2 内部组件，主要看第 4 个字母，例如 sqldInitDBCB 的第 4 个字母为 d，表

示数据管理。其他组件如图 9-2 所示。

Fourth Position Letters	Type of Activity Indicated
b	Buffer pool management and manipulation
c	Communications between clients and servers
d	Data management
e	Database engine processes
o	Operating system calls (such as opening and closing files)
p	Data protection (such as locking and logging)
r	Relational database services
s	Sorting operations
x	Indexing operations

图 92　DB2 常用底层组件

17 被调用函数返回的信息。可能会返回多个数据字段。

9.3　故障诊断工具

9.3.1　使用 db2support 收集环境信息

对 DB2 问题收集信息时，需要运行的最重要的 DB2 实用程序是 db2support。db2support 实用程序用于自动收集所有可用的 DB2 诊断信息和系统诊断信息，此外还有可选的交互式“问与答”会话，该会话会提出有关问题的详情。

如果使用 db2support 实用程序来帮助将信息传送至 IBM 支持机构，那么在系统遇到问题时运行 db2support 命令，该工具就会及时地收集信息，例如操作系统的性能详细信息。即便在出现问题时无法运行实用程序，也仍可以在问题停止之后发出 db2support 命令，因为会自动生成某些首次出现数据捕获(FODC)诊断文件，如图 9-3 所示。

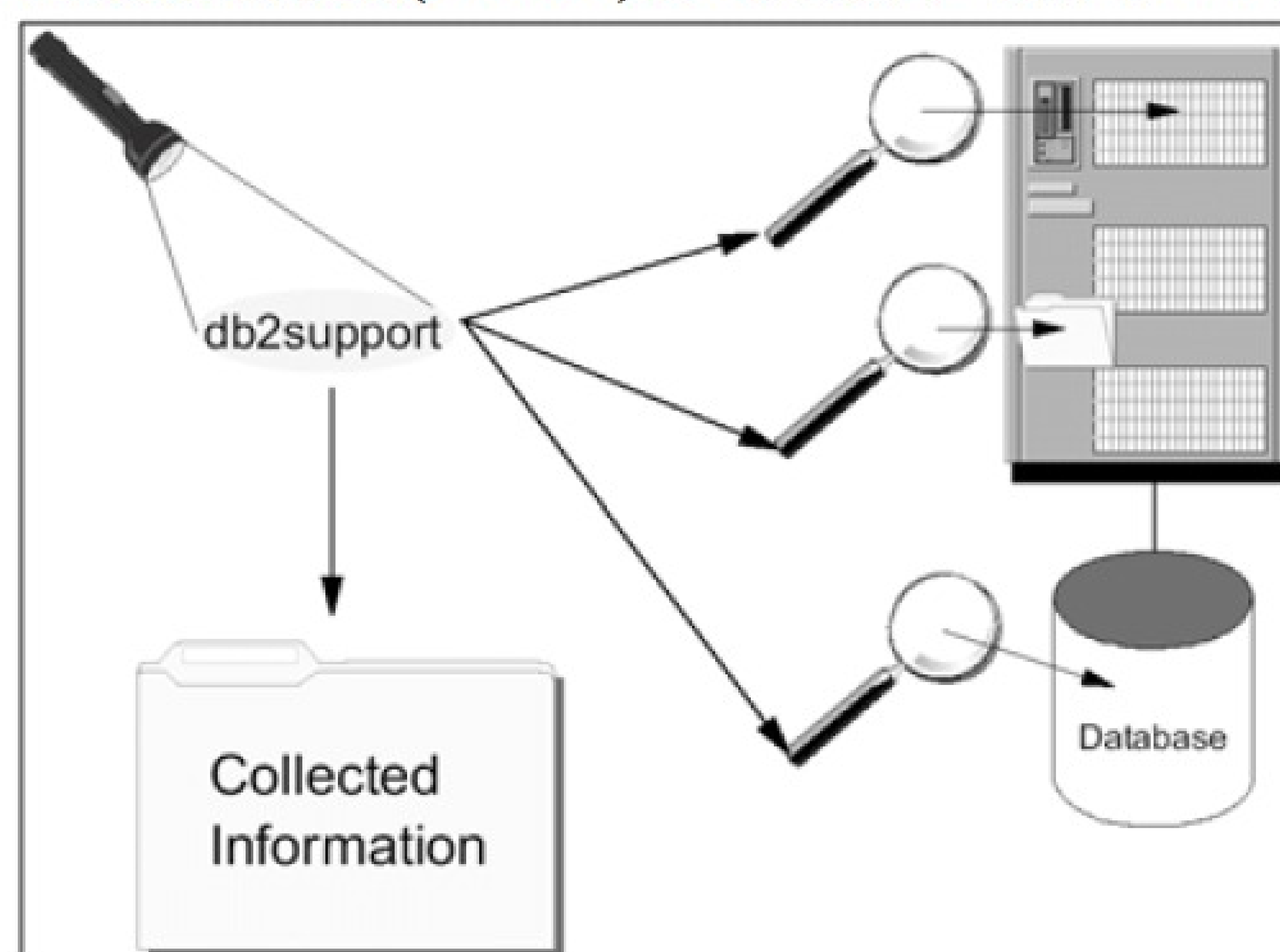


图 9-3　使用 db2support 收集环境信息

db2support 捕获的信息类型取决于调用命令的方式、是否启动了数据库管理器以及能否连接至数据库。对于调试问题所需的大多数信息的收集而言，以下基本调用通常已经足够(注意，如果使用-c 选项，那么该实用程序将建立与数据库的连接)：

```
    db2support <output path> -d <database name> -c
    db2support <directory> -d sample -s -m
```

输出的收集非常方便，并且会存储在压缩的 ZIP 归档 db2support.zip 中，以便可以很轻松地在任何系统上对其进行传送和解压缩。

db2support 命令收集到的默认信息包括有关操作系统、已安装 DB2 产品的版本和服务级别、数据库管理器和数据库配置的详细信息，以及有关以下各项的状态的信息：db2diag.log、所有陷阱文件、锁定列表文件、转储文件、各种与系统有关的文件、各种系统命令的输出、db2cli.ini、活动日志文件、缓冲池和表空间(SQLSPCS.1 和 SQLSPCS.2)控制文件、db2dump 目录的内容、扩展系统信息、日志文件头文件、恢复历史记录文件等几乎所有用到的信息。

9.3.2 db2ls 和 db2level

1. db2ls

由于能够在系统中安装 DB2 产品的多个副本，并且能够灵活地在你选择的路径中安装 DB2 产品和功能部件，因此需要使用工具来帮助跟踪已经安装了哪些 DB2 产品及其安装位置。在受支持的 Linux 和 UNIX 操作系统中(在 Windows 操作系统中不能使用 db2ls 命令)，db2ls 命令将列示安装在系统中的 DB2 产品和功能部件，可以使用 db2ls 命令来列示 DB2 产品在系统中的安装路径、DB2 产品级别、级别、修订包、特殊安装编号、安装日期、安装程序用户标识等。

db2ls 命令是可用来查询 DB2 产品的唯一方法。不能使用 Linux 或 UNIX 操作系统本机实用程序(例如 pkginfo、rpm、SMIT 或 swlist)查询 DB2 产品。包含用于查询 DB2 安装并与其进行交互的本机安装实用程序的任何现有脚本将需要进行更改。

要列示 DB2 产品在系统中的安装路径和 DB2 产品级别，请输入以下命令：

```
    # cd /usr/local/bin
    # ./db2ls
    Install Path   Level   Fix Pack   Special Install Number   Install Date
Installer UID
    /db2data/software/11.1.1 11.1.1.1 1  Wed Feb 22 14:15:09 2017 BEIST  0
```

上述命令将列出安装在系统中的每个 DB2 产品的下列信息。

要列出有关特定安装路径中 DB2 产品或功能部件的信息，必须指定 *q* 参数：

```
db2ls -q -p -b baseInstallDirectory
```

其中：

- *q* 指定要查询的产品或功能部件，此参数是必需的。*p* 指定列表只显示产品。
- *b* 参数指定产品或功能部件的安装目录。

```
#/usr/local/bin/db2ls -q -p -b /db2data/software/11.1.1

Install Path : /db2data/software/11.1.1

Product Response File ID   Level   Fix Pack   Product Description
DB2_SERVER_EDITION       11.1.1.1         1   DB2 Server Edition
```

2. db2level

db2level 命令帮助你确定 DB2 实例的版本和服务级别(构建级别和修订包级别)。

在 AIX 系统中运行 db2level 命令的典型结果为：

```
    $db2level
    DB21085I  This instance or install (instance name, where applicable:
"inst11")
    uses "64" bits and DB2 code release "SQL11011" with level identifier
"0202010F".
    Informational tokens are "DB2 v11.1.1.1", "s1612051900", "DYN1612051900AIX",
and Fix Pack "1".
    Product is installed at "/db2data/software/11.1.1".
```

这 4 个参考标记的组合唯一地标识 DB2 实例的精确服务级别。在定位问题以及联系 IBM 确认产品 bug 时，此信息非常重要。

对于 JDBC 或 SQLJ 应用程序，如果使用的是用于 SQLJ 和 JDBC 的 DB2 驱动程序，可通过运行 db2jcc 实用程序来确定驱动程序的级别：

```
    $ /usr/java6 64/bin/java -cp
/db2data/software/11.1.1/db2tss/lib/db2jcc4.jar com.ibm.db2.jcc.DB2Jcc
-version
    IBM Data Server Driver for JDBC and SQLJ 4.9.110
```

9.3.3 使用 db2diag 分析 db2diag.log 文件

db2diag 工具用于对 db2diag.log 中提供的大量信息进行过滤和格式化。过滤 db2diag.log 记录可缩短诊断问题时查找所需记录的时间。过去在没有这个命令之前，对 DBA 来说读取 db2diag.log 文件通常是非常痛苦的，因为很多时候客户没有清理 db2diag.log 文件，所以我们不得不频繁使用 grep 来过滤文件内容。下面举一些使用 db2diag 的示例。

例 9-1 按数据库名称过滤 db2diag.log。

如果实例中有若干数据库，并且只希望显示与数据库“SAMPLE”有关的消息，那么可以按如下所示过滤 db2diag.log：

```
    db2diag -g db=SAMPLE
```

因此，将仅显示包含“DB: SAMPLE”的 db2diag.log 记录：

```
    2016-09-14-18.43.33.495910+480 E16786A530        LEVEL: Event
    PID     : 15663356            TID  : 7755       PROC : db2sysc 0
    INSTANCE: db2inst1            NODE : 000         DB   : SAMPLE
    APPHDL  : 0-10                APPID: *LOCAL.DB2.120914104333
    AUTHID  : DB2INST1
    EDUID   : 7755                EDUNAME: db2stmm (SAMPLE) 0
    FUNCTION: DB2 UDB, Self tuning memory manager, stmmLogGetFileStats,
probe:565
    DATA #1 : <preformatted>
    New STMM log file (/home/db2inst1/sqllib/db2dump/stmmlog/stmm.0.log)
created automatically.
```

例 9-2 按进程标识过滤 db2diag.log。

以下命令可用来显示进程标识(PID)为 2200，并且在分区 0、1、2 或 3 上运行的进程生成的所有严重错误消息：

```
    db2diag -g level=Severe,pid=2200 -n 0,1,2,3
```

注意，此命令可能以不同方式写入，包括 db2diag -l severe -pid 2200 -n 0,1,2,3。还要注意的是，-g 选项指定区分大小写的搜索，所以此处“Severe”会起作用。但如果使用“severe”，将会失败。满足如下要求时，这些命令将成功检索 db2diag.log 记录：

```
    2016-02-13-14.34.36.027000-300 I18366H421        级别：严重
    PID     : 2200               TID  : 660        PROC : db2syscs.exe
    实例：DB2                    节点：000          数据库：SAMPLE
    APPHDL：0-1433               APPID：*LOCAL.DB2.060213193043
    函数：DB2 UDB，数据管理，sqldPoolCreate，探测点：273
```

```
返回码：ZRC=0x8002003C=-2147352516=SQLB_BAD_CONTAINER_PATH “错误的容器路径”
```

例 9-3　格式化 db2diag 工具输出。

以下命令过滤 2012 年 5 月 1 日之后发生，并且包含分区 0、1 或 2 上记录的所有非严重错误和严重错误的所有记录。它会输出匹配的记录，因此时间戳记、分区号和级别将出现在第一行上，而 pid、tid 和实例名将出现在第二行上，之后是错误消息：

```
$db2diag -t 2012-05-01 -n 0,1,2 -l severe,error
db2diag -fmt "Time: %{ts}
分区：%node Message Level: %{level} \nPid: %{pid}  Tid: %{tid}
实例：%{instance}\nMessage: @{msg}\n"
```

生成的输出示例如下所示：

```
时间：2012-05-15-19.31.36.099000 分区：000 消息级别：错误
Pid: 940 Tid: 40 实例：DB2
消息：ADM7506W 已经请求了数据库停顿
```

例 9-4　过滤来自不同工具的消息。

下列示例显示如何仅查看来自数据库管理器中特定工具(或所有工具)的消息。受支持的工具包括：

- ALL，这会返回来自所有工具的记录。
- MAIN，这会返回来自 DB2 常规诊断日志的记录，如 db2diag.log 和管理通知日志。
- OPSTATS，这会返回与优化器统计信息有关的记录。

要读取来自 MAIN 工具的消息，请使用以下命令：

```
db2diag -facility MAIN
```

要显示来自 OPSTATS 工具的消息并过滤出级别为 Severe 的记录，请使用以下命令：

```
db2diag -fac OPSTATS -level Severe
```

要显示来自所有可用工具的消息并过滤出实例为 harmistr、级别为 Error 的记录，请使用以下命令：

```
db2diag -fac all -g instance=harmistr,level=Error
```

要显示 OPSTATS 工具中级别为 Error 并且以特定格式输出时间戳记和 PID 字段的所有消息，请使用以下命令：

```
db2diag -fac opstats -level Error -fmt “ Time :%{ts} Pid :%{ts}”
```

例 9-5 下面的例子显示如何使用 db2diag 命令解释 DB2 跟踪或 db2diag.log 中的返回码。

```
$db2diag -rc fffffb95

Input ECF string 'fffffb95' parsed as 0xFFFFFB95 (-1131).
NOTE: /view/db2 v97fp5 aix64 s111017/vbs/engn/pd/../sqz/sqlzwhatisrc.C:
      V7 input ZRC 0xFFFFFB95 (-1131) may translate to V8 ZRC value of
0x83000B95 (-2097149035)

ZRC value to map: 0x83000B95 (-2097149035)
      V7 Equivalent ZRC value: 0xFFFFBB95 (-17515)

ZRC class :
      Unexpected Operating System error (Class Index: 3)
Component:
      Unknown component (Component Index: 0)
      Undefined as of DB2 v9.7.0.5; s111017
Operating system reason code (errno):
      0x00000B95 (2965) = Error 2965 occurred.
NOTE: Errno value 2965 is excessively large.
      It is possible that this error code is not a ZRC value.

Attempting to lookup value 0xFFFFFB95 (-1131) as a sqlcode
      Sqlcode -1131
SQL1131N  Stored procedure process has been terminated abnormally.
```

查找应用程序句柄 APPHDL 为 0-222 的所有诊断日志条目：

```
db2diag -g APPHDL="0-222"
```

查找应用程序句柄 APPHDL 为 0-222，并且在分区 0 上的所有诊断日志条目：

```
db2diag -g APPHDL="0-222",NODE=000
```

查找进程 1060946 的所有严重错误(Severe):

```
db2diag -g PID=1060946,LEVEL=Severe
```

查找所有 FUNCTION 名称中包含 fetch 的诊断日志条目：

```
db2diag -g FUNCTION:=fetch
```

查找所有组件名称以"base sys"开头的诊断日志条目：

```
db2diag -g "COMPONENT^=base sys"
```

查找所有返回码为"ZRC=0x80120086"的记录：

```
db2diag -g RETCODE:=0x80120086
```

除了过滤查找之外，db2diag 还可以格式化输出。可以指定查找结果的输出格式。关于格式化输出的详细帮助，请使用"db2diag -h fmt"命令查看。下面简单介绍一个例子：

```
    db2diag -time 2016-09-01 -node "0,1,2" -level "Severe, Error" |db2diag -fmt
"Time: %{ts} Partition: %node Message Level:%{level} \nPid: %{pid} Tid: %{tid}
Instance:%{instance}\nMessage: @{msg}\n"
```

该命令将查找 2012 年 9 月 1 日以来在分区 0、1、2 上错误级别为 Severe 和 Error 的错误，并按照下面的格式输出：

```
    Time : 2016-09-01-14.32.01.067843 Partition : 000 Message Level :Error
Pid : 1871948 Tid : 1 Instance :db2inst1
Message : ZRC=0x860F000A=-2045837302=SQLO FNEX "File not found."
DIA8411C A file "" could not be found.
```

db2diag 工具非常强大，有关更多信息，请发出下列命令：

- db2diag -help 提供所有可用选项的简短描述。
- db2diag -h brief 提供对所有不带示例的选项的描述。
- db2diag -h notes 提供用法说明和限制。
- db2diag -h examples 提供一小组示例以帮助入门。
- db2diag -h tutorial 提供所有可用选项的示例。
- db2diag -h all 提供最完整的选项列表。

9.3.4　db2pd 和 db2trc

1. db2pd

db2pd 是用于诊断和监视各种 DB2 数据库活动以及故障排除的监控工具，是从 DB2 V8.2 开始随 DB2 引擎发布的一款独立的实用程序，外观和功能类似于 Informix onstat 实用程序(其实就是 IBM 收购 Informix 后，DB2 从 Informix 数据库那里借鉴过来的)。并且随着 DB2 版本的不断更新，DB2 V9 和 DB2 V10 都对 db2pd 的功能进行了相关增强，比如，在 DB2 V9.7 修订包 1 和更高版本的修订包中，可以使用-reorgs 参数的 index 选项来显示有关索引重组的进度信息。db2pd 是从命令行以一种可选的交互模式执行的。该实用程序运行得非常快，因为不需要获取任何锁，并且在引擎资源以外运行(这意味着它甚至能在挂起的引擎上工作)。db2pd 功能非常强大，下面举一个使用 db2pd 生成堆栈跟踪的例子。

可使用 db2pd -stack all 命令来对当前数据库分区中的所有进程生成堆栈跟踪。如果怀疑某个进程或线程正在循环或正被挂起，那么可能要反复使用此命令。

可以通过发出命令 db2pd -stack <eduid>来获取特定引擎可调度单元(EDU)的当前调用堆栈。例如：

```
db2pd -stack 137
```

如果需要所有 DB2 进程的调用堆栈，请使用 db2pd -stack all 命令尝试转储实例的所有堆栈跟踪，生成文件在诊断路径下。例如：

```
$db2pd -stack all
Attempting to produce all stack traces for database partition.
See current DIAGPATH for stack trace files.
```

2. db2trc

db2trc 可以启动 DB2 的跟踪功能。所谓跟踪，实质上是运行跟踪程序时捕捉到的控制流(函数和相关参数值)的日志。对于 DB2 技术支持代表来说，如果只凭借错误信息返回的信息难于解决问题的话，那么跟踪就非常有用。

如果启动 DB2 跟踪功能，将会影响系统的性能，性能下降程度取决于在收集跟踪信息时要使用多少资源，应避免在生产环境中进行跟踪，而且应在 DB2 技术支持代表的指导下使用。在开发测试环境诊断问题时，可以尝试使用，通过分析跟踪信息可以知道故障时 DB2 的线程在执行哪些程序，从而判断出可能引起故障或性能问题的原因。但是由于抓到的信息涉及 DB2 的很多内部函数，因此一般的普通用户可能看不懂，这里的介绍仅供喜欢钻研的读者研究使用。

DB2 的跟踪信息中保存了目标线程或所有线程在某个时间段内程序的调用情况和执行状态。DB2 的跟踪信息可以通过 db2trc 或 db2pd 命令的特定选项获取到。db2trc 的简化语法如下：

```
db2trc on -p <pid.tid> -f <tid.trc>
30 秒或 10 秒或 5 秒
db2trc off
db2trc fmt -t <tid.trc><tid.fmt>
db2trc flw -t <tid.trc><tid.flw>
```

注意：

tid 指的是 keneltid，获得此信息需要在获得 eduid 后使用 db2pd –euds 进行转换。

9.3.5　DB2 内部返回码

有两种类型的内部返回码：ZRC 值和 ECF 值。这些返回码一般仅在供 IBM 软件支持机构使用的诊断工具中可视。例如，它们出现在 DB2 跟踪输出和 db2diag.log 文件中。

ZRC 和 ECF 值基本上具有相同的作用，但格式上稍有不同。每个 ZRC 值都具有以下特征： 类名、组件、原因码、相关联的 SQLCODE、SQLCA 消息标记、描述。ECF 值由以下部分组成：集名、产品标识、组件、描述。

ZRC 和 ECF 值通常为负数，并用于表示错误状况。ZRC 值根据它们表示的错误类型进行分组。这些分组称为“类”。例如，具有以“SQLZ_RC_MEMHEP”开头的名称的 ZRC 值通常是与内存不足相关的错误。相似地，ECF 值被分组为“集”。

以下是包含 ZRC 值的 db2diag.log 条目的示例：

```
    2016-08-04-18.02.43.071912+480 E3625592A820      LEVEL: Error (OS)
    PID     : 9830630             TID  : 6945       PROC : db2sysc 8
    INSTANCE: db2mds               NODE : 008
    APPHDL  : 0-5475
    EDUID   : 6945                EDUNAME: db2agntp 8
    FUNCTION: DB2 UDB, oper system services, sqloqualifydir, probe:20
    MESSAGE : ZRC=0x860F000A=-2045837302=SQLO FNEX "File not found."
            DIA8411C A file "" could not be found.
    CALLED  : OS, -, realpath
    OSERR   : ENOENT (2) "No such file or directory"
    DATA #1 : String, 12 bytes
    /db2/db2mdsc
    DATA #2 : unsigned integer, 8 bytes
    216
    DATA #3 : String, 0 bytes
    Object not dumped: Address: 0x07000000097FD9E8 Size: 0 Reason: Zero-length
data
    DATA #4 : String, 105 bytes
    Search for ossError*Analysis probe point after this log entry for further
    self-diagnosis of this problem
```

可使用 db2diag 命令获取有关此 ZRC 值的完整详细信息，例如：

```
    $db2diag -rc 0x860F000A
    $db2diag -rc 0x860F000A

    Input ZRC string '0x860F000A' parsed as 0x860F000A (-2045837302).

    ZRC value to map: 0x860F000A (-2045837302)
```

```
        V7 Equivalent ZRC value: 0xFFFFE60A (-6646)

ZRC class :
        Critical Media Error (Class Index: 6)
Component:
        SQLO ; oper system services (Component Index: 15)
Reason Code:
        10 (0x000A)

Identifer:
        SQLO FNEX
        SQLO MOD NOT FOUND
Identifer (without component):
        SQLZ RC FNEX

Description:
        File not found.

Associated information:
        Sqlcode -980
SQL0980C  A disk error occurred.  Subsequent SQL statements cannot be
processed.

        Number of sqlca tokens : 0
        Diaglog message number: 8411
```

如果发出命令 db2diag -rc -2045837302 或 db2diag -rc SQLO_FNEX，将返回相同信息。ECF 返回码的输出示例如下：

```
$db2diag -rc 0x90000076
$db2diag -rc 0x90000076

Input ECF string '0x90000076' parsed as 0x90000076 (-1879048074).

ECF value to map: 0x90000076 (-1879048074)

ECF Set :
        setecf (Set index : 1)
Product :
        DB2 Common
Component:
        OSSe
Code:
```

```
        118 (0x0076)

Identifier:
        ECF LIB CANNOT LOAD

Description:
        Cannot load the specified library
```

db2diag 命令输出中最有价值的故障诊断信息是描述和相关信息(仅对于 ZRC 返回码)。要获取 ZRC 或 ECF 值的完整列表，请分别使用命令 db2diag -rc zrc 和 db2diag -rc ecf。

9.4　故障诊断分析流程

9.4.1　故障诊断流程

图 9-4 列出了当数据库出现故障时的故障诊断检查列表。

DB2故障诊断检查列表

- [ ] 详细的故障描述：问题症状是什么？问题是在哪里发生的？问题何时发生的？发生问题的条件是什么？问题是否可以再现？
- [ ] 获取错误信息SQLCODE/SQLSTATE/Reason Code/操作系统错误日志判断是否存在硬件故障
- [ ] 结合操作系统判断是否有硬件故障和操作系统层面的软件错误
- [ ] 读取db2diag.log文件、dump文件、trap文件和db2cos脚本输出文件
- [ ] 操作系统版本补丁；利用db2ls和db2level判断DB2产品补丁信息
- [ ] 使用db2support收集故障信息；使用db2diag解释内部返回码，使用db2诊断工具来进一步定位问题
- [ ] 结合操作系统、应用和数据库综合判断：分析数据库配置参数，监控数据库运行情况
- [ ] 结合厂商判断是否是DB2数据库bug

图 9-4　DB2 故障诊断检查列表

可以看到，当系统出现故障时，为了准确地分析问题，第一步要做的就是完整地描述问题。如果没有问题描述，就不知道从什么地方开始调查造成问题的原因。此步骤包括询问自己如下基本问题：

- 症状是什么？
- 问题是在哪里发生的？
- 问题是在何时发生的？

- 发生问题的条件是什么？
- 问题是否可以再现？

通过回答上述及其他问题，就得到了对大多数问题的准确描述，并且也是找出问题解决方案的最好办法。

1. 症状是什么？

开始描述问题时，最明显的问题是“发生了什么问题？”。这看起来像一个很直观的问题；但是，它可以分为若干其他问题，从而更好地描述该问题。这些问题包括：

- 是什么人或什么工具报告该问题的？
- 错误代码和错误消息是什么？
- 怎么失败的？例如循环、中止、崩溃、性能下降或结果错误。
- 对业务有什么影响？

2. 问题是在哪里发生的？

确定问题发生的位置并不总是那么容易，但却是解决问题的最重要步骤之一。报告组件和失败组件之间可能存在多层技术。网络、磁盘和驱动程序只是调查问题时要考虑的几个部分。

- 是在特定平台上还是在多个平台上都发生了该问题？
- 是否支持当前环境和配置？
- 应用程序是在数据库服务器本地运行还是在远程服务器上运行？
- 是否涉及网关？
- 数据库是位于各个磁盘上，还是位于 RAID 磁盘阵列上？

这些类型的问题可帮助你隔离问题层，并且是确定问题来源所必需的。记住，不能只因为某层报告问题而总是断定那就是问题根源所在。

标识发生问题的位置时应了解发生问题的环境。总是应该花一些时间来完整描述问题环境，包括操作系统、操作系统版本、所有相应软件及版本和硬件信息。确认正在配置受支持的环境中运行，这是因为许多问题会解释为发现若干软件级别不能在一起运行，或者未在一起经过完整测试。

3. 问题是在何时发生的？

问题分析中的另一个必需步骤是创建导致故障的事件的详细时间线，对于从前发生的那些案例尤其如此。可以通过回溯工作过程很轻松地完成此任务：从报告错误时开始(尽可

能精确，甚至精确到毫秒)，然后通过可用日志和信息回溯工作过程。通常只需要进行观察，直到在任何诊断日志中发现的第一个值得怀疑的事件，但是，这并不总是那么容易，通常需要具有实践经验。如果同时存在多层技术，每层都有自己的诊断信息，那么要知道停止的时间就特别困难。

- 问题是否仅在日间或夜间的特定时间发生？
- 问题多久发生一次？
- 导致问题的事件的发生顺序是什么？
- 问题是否发生在环境改变(比如升级现有软件或硬件，或者安装新的软件或硬件)之后？

回答这类问题可帮助创建事件的详细时间线，并且提供用来进行调查的参考框架。

4. 发生问题的条件是什么？

要完整地描述问题，知道发生问题时还有什么别的软件在运行是非常重要的。如果问题是在特定环境或特定条件下发生的，那么这可能是找出问题原因的关键线索。

- 问题是否总是在执行同一任务时发生？
- 事件是否要按一定顺序发生，问题才会再现？
- 是否存在其他应用程序在同一时间失败？

回答这些类型的问题可帮助你说明发生问题的环境，并且能够将所有从属项关联起来。记住，不能只因为在同一时间发生了多个问题就表示它们总是相关的。

5. 问题是否可以再现？

从问题描述和调查角度来看，“理想”的问题是可以再现的。对于可再现的问题，几乎总是可以将它们与提供的一大堆工具或过程配合使用，以帮助进行调查。因此，可再现问题通常比较容易调试和解决。

但是，可再现问题也有缺点：如果该问题对企业有很严重的影响，那么可能不想让它再现。在这种情况下，最好尽可能在测试或开发环境中再现该问题。

- 能否在测试机器上再现问题？
- 是否多个用户或应用程序都遇到同一类型的问题？
- 能否通过运行单个命令、一组命令、特定应用程序或独立应用程序来再现问题？
- 能否通过从 DB2 命令行输入等价命令/查询来再现问题？

因为在调查时测试或开发环境有更好的灵活性，也更便于控制，所以最好在测试或开发环境中再现容易发生的单个问题。

9.4.2 结合系统事件判断

每个操作系统都有一组自己的诊断文件，用于跟踪活动和故障。最常见的(通常也是最有用的)诊断文件是错误报告或事件日志。

通常会忽略系统消息和错误日志。如果在问题定义和调查的初始阶段花时间执行一个简单任务，就可以在解决问题时节省几小时、几天甚至几星期的时间。该任务将会比较不同日志中的各个条目，并且记录看起来与时间和各个条目引用的资源相关的所有内容。

虽然并非总是与问题诊断有关，但许多情况下系统日志中会提供最好的线索。如果可将报告的系统问题与 DB2 错误关联起来分析，通常就能确定直接导致 DB2 症状的原因。很明显的示例包括磁盘错误、网络错误和硬件错误，并不那么明显的示例包括在不同机器上报告的错误，机器不同会影响连接时间或认证。

可以调查系统日志以评估稳定性，特别是在全新的系统中报告问题时尤其如此。在常用应用程序中间歇发生陷阱可能表示存在底层硬件问题。

以下是系统日志提供的一些其他信息。

- 重要事件，比如重新启动系统的时间。
- 系统中发生 DB2 陷阱(以及失败的其他软件中的错误、陷阱或异常)的时间。
- 内核应急启动、文件系统空间不足和交换空间不足错误(可能导致系统无法创建或派生新进程)。

系统日志可帮助你在 db2diag.log 中排除作为考虑因素的崩溃条目。如果在 DB2 管理通知或 DB2 诊断日志中发现崩溃恢复，但先前没有任何错误，那么 DB2 崩溃恢复可能是系统关闭造成的。以下是操作系统中的一些常见系统日志：

- AIX：使用/usr/bin/errpt -a 命令。
- Solaris：/var/adm/messages* 文件或/usr/bin/dmesg 命令。
- Linux：/var/log/messages* 文件或/bin/dmesg 命令。
- HP-UX：/var/adm/syslog/syslog.log 文件或/usr/bin/dmesg 命令。
- Windows：使用系统日志文件、安全性日志文件、应用程序事件日志文件以及 windir\drwtsn32.log 文件(其中，windir 是 Windows 安装目录)

9.4.3 结合系统运行状况诊断

在诊断与内存、交换空间、CPU、磁盘存储器和其他资源有关的一些问题时，需要完整地了解给定操作系统管理这些资源的方式。至少在确定与资源有关的问题时需要知道对于每个用户而言，该资源存在的限制及限制程度(相关限制通常用于 DB2 实例所有者的用户标识)。例如在 AIX 上，可以在/etc/security/limits 文件中设置 DB2 实例所有者 db2inst1

在内存 RSS、cpu、data、core、stack、文件系统 fsize 和 nofiles 等的资源使用限制。

以下是需要获取的一些重要配置信息：

- 操作系统补丁级别、已安装软件和升级历史
- CPU 数目
- RAM 大小
- 交换和文件高速缓存设置
- 用户数据和文件资源限制及每个用户的进程极限
- IPC 资源限制(消息队列、共享内存段和信号量)
- 磁盘存储器类型
- DB2 是否争用资源
- 认证在何处进行

大多数平台有直接的命令可用来检索资源信息。但是，你很少需要手动获取该信息，原因是 db2support 实用程序会收集此数据及更多其他信息。当指定-s 和-m 选项时，db2support 生成的 detailed_system_info.html 文件包含用来收集此信息的许多操作系统命令的语法。

9.5　案例分析

问题描述：

某系统在做 10 个进程并发压力测试的时候，发现运行 10 分钟左右，系统性能非常缓慢。初步定位为某 SQL 执行时间过长，耗时 30 秒钟，但是项目组分析后发现数据量很小，并且 SQL 很简单，并且完全命中索引。

问题分析：

(1) 登录数据库服务器，要求项目组重新开始压力测试，对数据库连接状态进行监控，发现压力开始几十秒钟后，所有代理都变为 Commit Active 状态，并且持续时间超过 30 秒，初步怀疑为写日志速度过慢造成的。

(2) 检查日志配置参数，发现日志缓冲区大小为 400KB，单个日志文件大小为 4MB，主日志只有 13 个，尝试将日志缓冲区和日志大小等调大之后重新测试，发现问题依然存在。

(3) 在出问题的时候对所有线程抓取堆栈：

```
$db2pd -stack all
```

分析日志，查找 Function 字段，发现所有线程都在等事务日志的写入：

```
    <StackTrace>
    -------Frame------ ------Function + Offset------
    0x09000000000EF1F8 thread_wait + 0x98
    0x0900000003B10F70 sqlpgWaitForLrecOnDisk  FP9SQLP DBCBUl + 0x24C
    0x0900000003B11A3C
sqlpWriteToLog__FP8sqeAgentUlN22P14SQLP LREC PARTP9SQLP LSN8 + 0x18
    0x0900000003B11C7C
sqlpWriteLR__FP8sqeAgentUlN42P14SQLP LREC PARTT2P9SQLP LSN8 - 0x6F0
    0x0900000003B36928 sqlptlog  FP8sqeAgentUlT2 + 0xD8
    0x0900000003B36EDC sqlpxcml  FP8sqeAgentP15SQLXA CALL INFOi + 0x210
    0x0900000003B3BD40 sqlrrcom dps  FP8sqlrr cbiT2P15SQLXA CALL INFO + 0x10C
    0x0900000003B3C7C4 sqlrrcom  FP8sqlrr cbiT2 + 0x30C
    0x0900000003B3CDF4 sqlrr commit  FP14db2UCinterface + 0x90
    0x0900000003B3D004 sqljs ddm rdbcmm  FP14db2UCinterfaceP13sqljDDMObject +
0x13C
    0x0900000003B44168
sqljsParseRdbAccessed  FP13sqljsDrdaAsCbP13sqljDDMObjectP14db2UCinterface +
0x4
    0x0900000003B43F20 sqljsParse  FP13sqljsDrdaAsCbP14db2UCinterface + 0x27C
    0x0900000003B45668 @63@sqljsSqlam  FP14db2UCinterfaceP8sqeAgentb + 0x180
    0x0900000003996B38 @63@sqljsDriveRequests  FP8sqeAgentP14db2UCconHandle +
0x98
    0x0900000003996980 @63@sqljsDrdaAsInnerDriver  FP18SQLCC INITSTRUCT Tb +
0xE0
    0x09000000039966E0 sqljsDrdaAsDriver  FP18SQLCC INITSTRUCT T + 0xD8
    0x0900000003998EF0 RunEDU  8sqeAgentFv + 0x124
    0x0900000003991F48 EDUDriver  9sqzEDUObjFv + 0x78
    0x0900000003A4E3F8 sqloEDUEntry + 0x57C
    </StackTrace>
```

(4) 怀疑文件系统出现了问题，于是检查系统系统日志，发现大量磁盘错误和链接错误：

```
    DCB47997   0428231712 T H hdisk5       DISK OPERATION ERROR
    DCB47997   0428231612 T H hdisk5       DISK OPERATION ERROR
    DCB47997   0428225912 T H hdisk2       DISK OPERATION ERROR
    DCB47997   0428225812 T H hdisk2       DISK OPERATION ERROR
    DCB47997   0428225712 T H hdisk2       DISK OPERATION ERROR
    DCB47997   0428225612 T H hdisk5       DISK OPERATION ERROR
    DCB47997   0428225612 T H hdisk2       DISK OPERATION ERROR
    DCB47997   0428225512 T H hdisk2       DISK OPERATION ERROR
    DCB47997   0428225412 T H hdisk2       DISK OPERATION ERROR
    DCB47997   0428225312 T H hdisk2       DISK OPERATION ERROR
    DCB47997   0428225212 T H hdisk2       DISK OPERATION ERROR
    DCB47997   0428224912 T H hdisk2       DISK OPERATION ERROR
```

```
DCB47997   0428224812 T H hdisk2        DISK OPERATION ERROR
4B436A3D   0428224812 T H fscsi0         LINK ERROR
4B436A3D   0428224812 T H fscsi0         LINK ERROR
DCB47997   0428224812 T H hdisk2        DISK OPERATION ERROR
4B436A3D   0428224812 T H fscsi0         LINK ERROR
DCB47997   0428224712 T H hdisk2        DISK OPERATION ERROR
DCB47997   0428224412 T H hdisk2        DISK OPERATION ERROR
DCB47997   0428224312 T H hdisk2        DISK OPERATION ERROR
DCB47997   0428224212 T H hdisk2        DISK OPERATION ERROR
DCB47997   0428224012 T H hdisk2        DISK OPERATION ERROR
DCB47997   0428223912 T H hdisk2        DISK OPERATION ERROR
DCB47997   0428223812 T H hdisk2        DISK OPERATION ERROR
DCB47997   0428223712 T H hdisk2        DISK OPERATION ERROR
DCB47997   0428223512 T H hdisk2        DISK OPERATION ERROR
```

```
#errpt -aj DCB47997

LABEL:          SC DISK ERR4
IDENTIFIER:     DCB47997

Date/Time:       Sun Apr 29 07:31:08 GMT+08:00 2016
Sequence Number: 8092
Machine Id:      000DBD2AD400
Node Id:         VIOS162 3
Class:           H
Type:            TEMP
WPAR:            Global
Resource Name:   hdisk5
Resource Class:
Resource Type:
Location:
VPD:
      Manufacturer................EMC
      Machine Type and Model......SYMMETRIX
      ROS Level and ID............5773
      Serial Number...............76666390
      Part Number.................000000000000510026000190
      EC Level....................105676
      LIC Node VPD................0666
      Device Specific.(Z0)........00
      Device Specific.(Z1)........51
      Device Specific.(Z2)........000000300D9C09000260019C
      Device Specific.(Z3)........12000000
```

```
        Device Specific.(Z4)........5410D9A4
        Device Specific.(Z5)........FF80
        Device Specific.(Z6)........4D
    Description
    DISK OPERATION ERROR
    Probable Causes
    MEDIA
    DASD DEVICE
    User Causes
    MEDIA DEFECTIVE

        Recommended Actions
        FOR REMOVABLE MEDIA, CHANGE MEDIA AND RETRY
        PERFORM PROBLEM DETERMINATION PROCEDURES

    Failure Causes
    MEDIA
    DISK DRIVE

        Recommended Actions
        FOR REMOVABLE MEDIA, CHANGE MEDIA AND RETRY
        PERFORM PROBLEM DETERMINATION PROCEDURES

    Detail Data
    PATH ID
             0
    SENSE DATA
    0A00 2A00 0104 1DC0 0000 2804 0000 0000 0000 0000 0000 0000 0200 0300 0000
0000
    0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000
    0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000
    0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000
    0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000
    0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000
    0000 0000 0000 0000 0686 0007 0680 0000 0000 0000 0000 0000 0000 0000 0083
0000
    0000 0035 001D
```

(5) 之后联系系统工程师，在解决链路报错的问题后，重新开始压力测试，性能恢复正常。

9.6　本章小结

本章讲解了关于故障诊断的一些知识，其实在复杂的应用环境中，故障诊断特别复杂。而且 DB2 相对来说提供的故障诊断日志稍显封闭(相对于 Oracle)，所以很多问题的诊断必须建立在对 DB2 内部体系结构非常了解的基础上才能做出准确的判断。

第10章 DB2 案例精选

本章我们对日常生活中常见的 DB2 问题进行总结，希望这些总结能够帮助你用好 DB2 数据库。

10.1 实例常见问题和诊断案例

10.1.1 实例无法启动问题总结

db2nodes.cfg 这个配置文件主要是为了数据库分区设计的，下面我们来看看这个文件的内容：

```
    test2:/home/db2inst4/sqllib$cat db2nodes.cfg
    0 test2 0
```

在这个文件中，最重要的是 test2，这是我机器的主机名。是我创建实例的时候，DB2 自动读取机器主机名然后写到这个文件中的。所以你如果更改了机器的 hostname，而没有更改这个配置文件，那么你的实例无法启动。在我更改了机器的主机名后，启动实例时报如下错误：

```
    $db2start
    SQL6048N  A communication error occurred during START or STOP DATABASE
MANAGER processing.
```

所以，如果你的实例无法启动，请检查 db2nodes.cfg，看该文件配置是否正常。

在确保实例的 db2nodes.cfg 文件配置正常后，如果启动实例仍然报错，这个时候请检查 db2systm 实例配置文件。实例启动时需要读取这个文件来分配资源，如果这个文件被破坏，实例将无法正常启动。

在 HP-UX、Solaris 和 Linux 上安装 DB2 时需要设置相关操作系统内核参数(共享内存、信号灯、消息队列等)。如果内核参数设置不当，也会导致实例无法启动。

上述几个是实例无法启动的主要问题，当然还有其他很多原因，但是不太常见。例如：如果更改实例目录中 security 目录下文件的属组和权限，或者数据库文件权限被改成了777，实例可能也无法启动。当然这种情况很难检查，也不太常见。

10.1.2 实例无法正常终止

某些情况下，数据库异常。我们需要停止实例，但是发出 db2stop 或 db2stop force 后，一直 hang 在那里无法正常终止。这种情况下我们只能通过如下方法来停止实例：

- 执行 db2_kill 或 ps –efl |awk "{print $2 }"| xargs kill -9
- 执行 ipcs -a | grep -i db2inst1(你机器上的实例名)，查看系统为该实例分配的共享内存
- 执行 ipcrm -m ID 来清除实例内存段或 ipclean
- 执行 ipcs | grep <instancename> | awk '{print " ipcrm -"$1" "$2 }' > ipcln
- 执行 chmod +x ipcln
- 执行 ipcln
- 在窗口中执行 db2nkill，然后停止服务，实在不行重启机器。

一般情况下，实例无法正常终止通常和进程异常退出或应用程序有关。用这种方式可以把实例停止，至于为什么实例无法正常终止，这个原因太多。

10.1.3 实例目录误删除

如果实例目录不小心被误删除，并且是双机热备环境，那么建议您切换到备机上，获取实例的 DBM 配置文件。然后执行：

```
db2 list db directory
```

查看数据库的目录，假如数据库的目录为“/db2prod/data”。

在主机上，执行 db2idrop 删除实例，然后执行 db2icrt 重新创建实例。实例创建后，根据备机的 DBM 配置文件把配置文件同步。最后在该实例上编目数据库：

```
db2 catalog db mydb on /db2prod/data
```

10.1.4 实例崩溃问题

遇到实例崩溃的问题，首先查看 db2diag.log，根据里面的信息来分析数据库宕机的原因。再看 db2dump 目录中是否产生 trap 文件。可以根据这些信息来分析原因，一般这类问题都需要 IBM 工程师协助解决。宕机的原因可以分为两类，一类是数据库的 bug，是由数据库的缺陷引起的，如果遇到数据库的缺陷，都有临时的解决方案，或者通过安装最新的补丁来解决，对某些问题 IBM 也提供临时的修订来解决(需要付费)。另一类是操作系统误操作等非产品问题导致的，对非产品问题导致的宕机尽量要避免。常见的实例宕机原因：

- 系统交换空间(paging space)用尽
- 数据库的核心进程被 kill
- 应用进程异常退出且未释放资源

10.2 数据库常见问题总结

10.2.1 数据库日志空间满 SQL0964C 错误

数据库运行时如果报 SQL0964C 错误，一般和数据库日志有关，先看下面的案例：

```
    $db2 "import from data.del of del insert into t1"
    SQL3109N  The utility is beginning to load data from file "data.del".
    SQL3306N  An SQL error "-964" occurred while inserting a row into the table.
    SQL0964C  The transaction log for the database is full.  SQLSTATE=57011
    SQL3110N  The utility has completed processing.  "709" rows were read from
the input file.
```

DB2 使用的活动日志的最大空间是由下面公式：

(logprimary + logsecond) * logfilsiz * 4096KB

计算出的大小决定的(logprimary、logsecond 和 logfilsiz 是数据库配置参数)。在 DB2 中，一个长事务最多可以使用不超过 1024GB 的日志(DB2 V9)。

当空间已全部被分配，而应用仍试图请求更多活动日志空间时，就会发生日志满的情况。此时，用户的更新、删除或插入操作都会使 DB2DIAG.LOG 中写入以下信息：SQL0964C 数据库的事务日志已满。

SQL0964C 错误表明数据库日志已满，DB2 活动日志满通常是由于存在大量未提交事

务的数据，使得活动日志的空间不能及时释放，使新的事务无法申请到可用日志空间而最终报出 SQL0964C 错误所致。如果要解决这些问题，你可以增加日志文件或日志个数。这种错误一般是批量插入、批量删除或批量更新时报的错误。所以最好是能够调整业务逻辑，分批次删除、更新或插入。这样做会降低日志报错的概率。

但还有另外一种原因，即在日志空间并未用尽的情况下，若某个占有最旧活动日志的应用长时间未做提交操作，阻止了日志的 LSN 的分配，造成日志空间无法使用，同样会引发这一日志满的报错。对于这种情况，可以提交事务或利用 FORCE 命令来终止此应用程序，以便释放它所占用的日志空间，使 LSN 可以继续分配、空闲的日志空间可用。这里就提供了由这一原因导致日志满问题的解决方法。

首先检查 DB2 诊断日志文件 db2diag.log，在其中查找类似如下信息：

```
2016-01-16-02.53.54.935308 Instance:db2inst1 Node:016
PID:144252(db2agntp (SAMPLE) 16) Appid:*.*
data protection sqlpgrsp Probe:50 Database:SAMPLE
Log Full -- active log held by appl. handle 787273
End this application by COMMIT, ROLLBACK or FORCE APPLICATION.
```

由此，可以找到最早持有日志空间的应用程序，其句柄为 787273。使用 DB2 的快照 db2 get snapshot for db on <dbname>|grep –i old，通过从快照的输出中查找类似以下信息：

```
Appl id holding the oldest transaction = 787273
```

同样可以找到这个应用程序的句柄。这时使用以下命令可以在无须断开数据库与其他应用程序连接的情况下强行终止该应用程序：

```
db2 force application (787273)
DB20000I FORCE APPLICATION 命令成功完成。
DB21024I 该命令为异步的，可能不会立即生效。
```

根据提示，由于该命令是异步操作，可再次使用：

```
db2 get snapshot for application agented 787273
```

验证应用是否已被真正停止。如果该应用的状态为 ROLLBACK，请等待回滚完毕后变为 UOW WAITING 就会释放日志空间，而 DB2 日志因此重新可用。

10.2.2 数据库时区和时间

在数据库创建好之后，调整系统时间会造成数据库内部时间戳的异常。数据库中的一些对象和时间相关，一旦时间不准确要调整，需要很小心。错误的时间调整可能会造成很

多问题，比如：

- 某些对象失效，例如 SQL0440N，找不到具有兼容自变量的类型为“<例程类型>”的名为“<例程名>”的已授权例程。
- 数据库日志逻辑错误->宕机。
- 常见错误->只调整时间，未调整时区。

正确的做法是在数据库创建之前，调整好时间和时区。如果在数据库创建好之后，确实需要调整时间和时区，建议停止实例，然后正确地调整时间和时区。

10.2.3　中文乱码和代码页转换

当执行命令 db2 connect to sample 后，系统返回如下错误：

```
SQL0332N There is no available conversion for the source code page "819"
to the target code page "1386". Reason Code "1". SQLSTATE=57017
```

此为 DB2 要求源代码页与目标代码页是彼此兼容的。在上述例子中，源代码页为 819，目标代码页为 1386，两者不兼容才导致了数据库连接失败。解决方法如下：请在执行 db2 connect 命令失败的机器上，开启 DB2 命令窗口并在执行以下命令后重新进行连接。

```
db2set db2codepage=819
```

由于使用单字节字符集码页创建的数据库来存放双字节字符的方式不为 DB2 正式支持，因此这里介绍的方法只能帮助用户在 DB2 命令编辑器中正确显示库中的双字节字符，但在其他 Java 应用中，这样的数据库仍然可能会遇到同样的问题，而且代码页在创建数据库时一旦选择，以后就无法更改代码页，所以我们还是强烈建议选择相应的双字节字符集码页来创建数据库。你也可以在创建数据库时选择 UTF(unicode)方式来创建数据库以避免这一类问题。

总之，所有的代码页问题基本上都是因为当前运行环境代码页和数据库服务器代码页不一致造成的。如果在二者一致的情况下数据显示不正常，通常与创建数据库时指定单字节(SBCS，欧美是单字节)和双字节(DBCS，东亚国家是双字节)有关。

10.2.4　通信错误 SQL30081N

请看下面的例子：

```
db2 connect to sample user Informix using Informix
SQL30081N  检测到通信错误。正在使用的通信协议："<协议>"。正在使用的通信 API："<接口
```

```
>"。检测到错误的位置："<位置>"。检测到错误的通信功能："<功能>"。特定于协议的错误代码：
"<rc1>"、"<rc2>" 和"<rc3>"。
```

一般出现这种错误时，请按照以下步骤检查：

(1) 在客户机上用 ping 或 pctt 命令连接数据库服务器，确保连接成功。

(2) 用 netstat 命令检查实例侦听端口，确保实例端口处于“listen”或”listening”状态。

(3) 检查实例配置文件 SVCENAME，确保设置正确。

(4) 检查 db2set –all 输出，确保 DB2COMM 注册变量的通信协议正确设置。

(5) 检查/etc/services 文件，确保实例端口所在行没有使用 TAB 空格，以及 windows 上最后一行是回车。

有的时候，客户机的防火墙也会造成 SQL30081N 通信错误。

10.2.5 数据库备份、前滚暂挂

当数据库从循环日志改成归档日志时，数据库要求进行一次脱机备份，在重新启动数据库后，数据库就处于备份暂挂(backup pending)状态。如果要消除备份暂挂状态，必须对数据库做完整备份。

当我们执行数据库的前滚恢复时，如果没有指定 without rolling forward，那么恢复后的数据库会处于 rollforward pending 状态。在这种情况下，只有对数据库前滚才能访问数据库。

总之，暂挂(pending)是 DB2 数据库的一种内部保护完整性的一种机制，数据库处于什么样的暂挂(pending)状态，我们就执行相应的什么样的操作。

10.2.6 数据库活动日志删除

DB2 数据库的启动环节中有一个步骤是检查 SQLOGCTL.LFH.1 文件，这个文件中记录数据库的活动日志情况。很多人由于没有深刻了解活动日志的作用，因此有可能误删除活动日志。如果误删除活动日志，那么数据库将无法连接。这也属于数据库损坏的一种情况，因为数据库的完整一致性受到了破坏。出现这种情况，首先考虑是否有可以恢复的备份，如果有，可以从备份恢复，然后前滚到日志的末尾以完全恢复数据库。如果没有可用的备份来恢复，可以通过 IBM 的技术支持中心来协助解决(IBM 内部通过一条命令 db2lfh 去修改 SQLOGCTL.LFH.1 文件)。如果想自己解决，那只有使用 db2dart 工具了。

如何避免数据库的活动日志被删除？可以有以下措施：

- 启用数据库的镜像日志功能，但是会带来性能的额外开销。
- 当一定要手工删除活动日志目录中的归档日志时，使用命令 PRUNE LOGFILE PRIOR TO log-file-name，可以避免将活动日志误删除。

- 注意维护活动日志所在目录的权限。

10.2.7　数据库损坏(数据页、索引页)SQL1043C

下面我们先来看一个实际的案例，请看下面的 db2diag.log 文件：

```
    2011-09-22-11.46.45.864000+480 I805726H366 LEVEL: Error
    PID : 1860 TID : 2732 PROC : db2syscs.exe
    INSTANCE: DB2INST NODE : 000
    FUNCTION: DB2 UDB, buffer pool services, sqlbReadAndReleaseBuffers,
    probe:13
    RETCODE : ZRC=0x86020001=-2046689279=SQLB BADP "page is bad"
DIA8400C A bad page was encountered.
2011-09-22-11.46.45.910000+480 I806094H413 LEVEL: Error
PID : 1860 TID : 2732 PROC : db2syscs.exe
INSTANCE: DB2INST NODE : 000
FUNCTION: DB2 UDB, buffer pool services, sqlbReadAndReleaseBuffers, probe:13
DATA #1 : String, 126 bytes
Obj={pool:34;obj:6;type:0} State=x27 Page=140354 Cont=0 Offset=140352
BlkSize=12
sqlbReadAndReleaseBuffers error: num-pages=8

2011-09-22-11.46.45.942000+480 I806509H593 LEVEL: Error
PID : 1860 TID : 2732 PROC : db2syscs.exe
INSTANCE: DB2INST NODE : 000
MESSAGE : SQLB OBJECT DESC
DATA #1 : Hexdump, 68 bytes
0x04B6B5DC : 2200 0600 2200 0600 0000 0000 003A A2A6 "..."........:..
0x04B6B5EC : 40E5 0000 0000 0000 0000 0000 0000 0000 @...............
0x04B6B5FC : 0000 0000 0101 0000 2700 0000 0000 0000 ........'.......
0x04B6B60C : 0010 0000 2000 0000 0100 0000 2200 0600 .... ......."...
0x04B6B61C : 408C 7400 @.t.
SQL1034C The database is damaged.
2011-09-22-11.46.46.020000+480 I807104H356 LEVEL: Error
PID : 1860 TID : 2732 PROC : db2syscs.exe
INSTANCE: DB2INST NODE : 000
FUNCTION: DB2 UDB, buffer pool services, sqlbErrorHandler, probe:0
RETCODE : ZRC=0x86020001=-2046689279=SQLB BADP "page is bad"
DIA8400C A bad page was encountered.

2011-09-22-11.46.46.020000+480 I807462H351 LEVEL: Error
PID : 1860 TID : 2732 PROC : db2syscs.exe
INSTANCE: DB2INST NODE : 000
```

```
FUNCTION: DB2 UDB, buffer pool services, sqlbErrorHandler, probe:0
DATA #1 : String, 75 bytes
Obj={pool:34;obj:6;type:0} State=x27
Prefetcher Error, in sqlbProcessRange

2011-09-22-11.46.46.020000+480 I807815H593 LEVEL: Error
PID : 1860 TID : 2732 PROC : db2syscs.exe
INSTANCE: DB2INST NODE : 000
MESSAGE : SQLB OBJECT DESC
DATA #1 : Hexdump, 68 bytes
0x04B6B5DC : 2200 0600 2200 0600 0000 0000 003A A2A6 "..."........:..
0x04B6B5EC : 40E5 0000 0000 0000 0000 0000 0000 0000 @...............
0x04B6B5FC : 0000 0000 0101 0000 2700 0000 0000 0000 ........'.......
0x04B6B60C : 0010 0000 2000 0000 0100 0000 2200 0600 .... ......."...
0x04B6B61C : 408C 7400 @.t.
```

“Obj={pool:34;obj:6;type:0} State=x27”中的 pool 指的是表空间 ID，obj 指的是对象 ID，从系统表中读取判断是哪个表受到损坏。例如：

```
select tabname from syscat.tables where tbspaceid=4 and tableid=6
```

数据库最严重的故障莫过于数据库损坏，从上面的例子来看，我们的数据库中有数据页受到损坏。出现 SQL1034C 时，我们首先用操作系统命令，例如在 AIX 操作系统上：使用 errpt –d H –T PERM 来判断系统是否出现硬件损坏。然后尝试能否使用 db2 restart db sample 命令让数据库执行崩溃恢复。

如果上述办法都不能解决问题，最好的办法是从备份恢复数据库。如果无法从备份恢复，可以根据损坏的原因尝试相应的解决方案。由于存储问题导致部分数据文件损坏，但是数据库还可以连接，这种情况可以采用导出数据库的表结果和数据的方法来恢复数据库。当然对损坏的表，导出是无法完成的，这时可以使用 db2dart 的导出数据功能来导出这些损坏的表中的数据。如果数据库损坏到已经无法连接的程度，那么除了从备份恢复，唯一的办法是使用 db2dart 来导出所有的数据。

```
运行命令 db2dart /DDEL
    # Table object data formatting start.
    # Please enter
    # Table ID or name, tablespace ID, first page, num of pages:
    # (suffic page number with 'p' for pool relative),
```

按照提示输入表名、表空间 ID、起始页数、需要导出的页数。如果你的数据库非常大，

这将是一件工作量非常大的事情。所以建议大家做好数据库备份。

10.2.8 索引重新构建问题

当数据库中的索引无效时，可以使用 DBM 配置参数 INDEXREC 决定索引重新构建的方式。

```
$ db2 get dbm cfg | grep -i indexrec
索引重新创建时间                                   (INDEXREC) = ACCESS
```

此参数指示数据库管理器何时将尝试重建无效的索引，以及在 DB2 前滚期间或在辅助数据库上重放 HADR 日志期间是否重做任何索引构建。

- SYSTEM 在数据库管理器配置文件中指定的 use system setting 决定何时将重建无效的索引。
- ACCESS 第一次访问索引时将重建无效的索引。
- ACCESS_NO_REDO 第一次访问基础表时将重建无效的索引。
- RESTART indexrec 的默认值：将在显式地或隐式地发出 RESTART DATABASE 命令时重建无效的索引。
- RESTART_NO_REDO 将在显式或隐式地发出 RESTART DATABASE 命令时重建无效的索引。

建议：在高端用户服务器上，如果重新启动所花费的时间不重要，此选项的最佳选择将是在数据库重新启动时重建索引，以作为在崩溃后重新将数据库联机的过程的一部分。将此参数设置为“ACCESS”或“ACCESS_NO_REDO”将导致重新创建索引时数据库管理器的性能降低。任何访问该特定索引或表的用户将不得不等待，直到索引被重新创建为止。如果将此参数设置为“RESTART”，重新启动数据库所花费的时间将因重新创建索引而延长，但是一旦数据库恢复联机，正常处理将不受影响。读者应该了解这几种方式的区别以确定你的数据库选用那种方式是最合理的。

10.2.9 DB2 实用程序不可用

通常在我们做完 DB2 升级或打完补丁以后，我们在执行数据库的一些命令或实用程序时常常报一些错误，例如：

```
SQL0805N Package “NULLID.DYNEXPLN 0X5142316d564fa32” was not found
```

等类似错误，对这种问题我们按照以下方法解决：进入$DB2HOME/sqllib/bnd 目录。执行如下命令：

```
cd /home/db2inst1/sqllib/bnd
db2 connect to sample
db2 bind @db2ubind.list isolation cs blocking all grant public
-----重新绑定应用程序绑定文件
```

10.2.10　快速清空表数据

对于使用 DB2 数据库的用户，有时候需要将表中数据清空，这里提供了 4 种数据删除的方法，以供用户根据自己的需求进行选择：

- 使用 DELETE 语句

```
DELETE FROM <表名>
```

该语句将清除表中所有数据，但由于这一操作会记日志，因此执行速度会相对慢一些。另外要注意的是，如果表较大，为保证删除操作成功，应考虑是否留有足够大的日志空间。

- 使用 NOT LOGGED INITIALLY 选项

```
ALTER TABLE <表名> ACTIVATE NOT LOGGED INITIALLY WITH EMPTY TABLE
```

这一方法仅在所操作的表在创建时选择 NOT LOGGED INITIALLY 选项进行定义后才可使用。整个删除操作将不会记日志，因此执行速度是几种方法中最快的一种，但删除的数据是不可恢复的。

- 使用 LOAD 命令

```
LOAD FROM /dev/null OF DEL REPLACE INTO <表名> NONRECOVERABLE --(Unix 系统)
或 LOAD FROM <空文件> OF DEL REPLACE INTO <表名> NONRECOVERABLE
```

在这一方法中，REPLACE 导入方式首先会将表中所有数据清空，然后 IMPORT/LOAD 又向表中导入了空数据，从而实现了数据的清除操作。

- 使用 DROP/CREATE TABLE 语句

```
DROP TABLE <表名>
CREATE TABLE <表名> <字段的定义>
```

如果保存了表的定义语句，或已利用 db2look 命令获得了表定义的脚本，那么也可先删除整个表，再重新创建表。如果表较大，用这种方法实现数据清空的速度会快于使用 DELETE 语句。但是用这种方法要注意：如果这个表上有很多外键，那么需要维护外键的完整性。

10.2.11　表和索引统计信息不一致

当表正在被更新的时候，可以对其执行 RUNSTATS 命令，进行表和索引数据统计信息的收集，但根据更新操作级别的不同，得到的统计信息可能是不一致的。您可能会遇到如下错误信息：

```
    SQL2314W 某些统计信息处于不一致的状态。最近收集的 "<对象-1>"统计信息与现有的 "<对象-2>" 统计信息不一致。
```

统计信息不一致可能会导致不理想的查询计划，SQL2314W 就是产生这种可能性的警告信息。您应该尝试在应用对表的访问级别尽可能低(或者如果可能的话没有任何操作)的情况下执行 RUNSTATS 命令，比如尝试尽量避免在有更新操作的情况下进行 RUNSTATS 操作。

另外，RUNSTATS 命令默认使用的是“ALLOW WRITE ACCESS”选项，您也可以使用选项“ALLOW READ ACCESS”来执 RUNSTATS。这样一来，在 RUNSTATS 执行的时候，其他操作将不能更改表。但这个选项会对应用的并行性有影响，因为任何想要更改表的操作都会处于等待状态。为了减少表被 ALLOW READ ACCESS 选项的 RUNSTATS 锁定的时间，您可以考虑使用 TABLESAMPLE 选项，这个选项导致 RUNSTATS 对于表的部分采样数据而不是对所有数据收集统计信息。对于采样数据大小的合理选择，可以在确保统计信息一致性的情况下，加快 RUNSTATS 的速度。

如果以上建议都无法阻止 SQL2314W 警告信息的出现，而检查访问表的应用的存取计划时发现确实存在优化器未能自动选择最优存取计划的情况，应考虑在尽量保证 RUNSTATS 可获得较高存取权限的时候重新执行 RUNSTATS，以便优化器重新产生最优的存取计划。不过对于因遇到 SQL2314W 产生的非最优的存取计划，比如本应选择索引扫描，但优化器选择了表扫描的情况，也可以考虑用 ALTER TABLE 语句将表标记成“volatile”，以鼓励优化器选择索引扫描而不考虑表扫描。

10.3　表空间状态

DB2 用表状态和表空间状态来控制对数据的访问，或者在特定情况下帮助保护数据库的完整性。下面我们总结表空间和表中可能引出特定状态的更常见的一些条件，您可以用它们来识别哪些状态是有效的，以及如何做出正确响应，以便可以继续使用数据。理解这些状态还可以使我们更好地理解数据库的行为。

表空间状态在某些情况下被用来控制对数据的访问，或者在必要时被用来引出特定用户动作以保护数据库的完整性。大多数状态产生于与某个 DB2 实用程序的操作相关的事

件，例如加载实用程序、备份和恢复实用程序。下面我们举一些例子，以便准确地展示如何解释和响应管理数据库时可能碰到的状态。

db2tbst 命令接收十六进制的状态值，并返回相应的表空间状态(参见图 10-1)。例如，命令 db2tbst 0x0008 返回 State=load pending，而十六进制的状态值反过来又是 LIST TABLESPACES 命令输出的组成部分(参见图 10-2)。

```
▶▶—db2tbst—tablespace-state————————————————————————————▶◀
```

图 10-1　db2tbst 命令

表空间的外部可见状态是由单个状态值的十六进制总和构成的。例如，如果表空间的状态是 backup pending 和 load in progress，那么返回的十六进制值就是 0x20020 (0x00020 + 0x20000)。在本例中，命令 db2tbst 0x20020 返回：

```
State = Backup Pending
     + Load in Progress
```

```
▶▶—LIST TABLESPACES—┬─────────────┬——————————————————————▶◀
                    └─SHOW DETAIL─┘
```

图 10-2　可以使用 LIST TABLESPACES 命令确定连接数据库中表空间的当前状态

下面我们讲解一些最经常碰到的表空间状态。

10.3.1　backup pending

在执行指定时间点(point-in-time)的表空间前滚操作之后，或者在执行指定了 COPY NO 选项的 LOAD 操作(针对可恢复的数据库)之后，表空间处于这种状态。在使用表空间之前，必须备份表空间(或是整个数据库)。如果没有备份这个表空间，那么只能对其中包含的表进行查询，而无法更新它们。注意：在启用数据库进行前滚恢复之后，还必须立即对数据库进行备份。如果开启了 LOGARCHMETH1 ，即把 LOGARCHMETH1 的值从 OFF 改为一个路径，那么数据库会变为 backup pending。直到对这样的数据库进行了备份，您才可以连接它。备份后，backup_pending 数据库配置参数会被设为 NO。

已知载入的输入文件 staff_data.del 具有以下内容：“11，"Melnyk"，20，"Sales"，10，70000，15000”。

```
update db cfg for sample using LOGARCHMETH1 disk:/logs;
---------此时数据库处于 BACKUP  PENDING 状态
```

10.3.2　脱机(offline and not accessible)

如果表空间的一个或多个容器存在问题，那么表空间就处于脱机(offline and not accessible)状态。容器偶然可能会被重命名、移动或损坏。在该问题被纠正，并且再次可以访问与表空间相关的容器之后，可以通过断开数据库与应用程序的连接，然后重新连接数据库来消除该异常状态。或者，您可以执行一条 ALTER TABLESPACE 语句，指定 SWITCH ONLINE 子句来消除表空间的 offline and not accessible 状态，从而无须断开其他应用程序与该数据库的连接：

```
    connect to sample;
    create tablespace data space managed by database using (file
'c:\prod\data container1' 1024);
    -----模拟容器故障，手工执行操作系统 mv 或 rename，表空间容器为
'c:\prod\data container2'。
    select * from staff---------此时数据库处于 Offline and Not Accessible 状态
```

该查询返回 SQL0290N(不允许访问表空间)，而 LIST TABLESPACES 命令返回 TS1 的状态值 0x4000(offline and not accessible)。将表空间容器'c:\prod\data_container2'重命名为'c:\prod\data_container1'。这一次，该查询将运行成功。

什么情况下会处于 offline 状态呢？下面举一个实际生产中的例子。在双机热备 HA 的环境中，客户在主机上重新创建了使用裸设备的表空间后，未同步 HA 环境。结果导致主机故障，在切换到备机时，由于裸设备权限不正确而导致表空间处于 offline 状态。

10.3.3　quiesced exclusive | share | update

当调用表空间停顿(quiesce)功能的应用程序独占(读或写)、共享或意向更新访问表空间时，表空间就处于这种状态。您可以通过执行一条 QUIESCE TABLESPACES FOR TABLE 命令，将表空间置于 quiesced exclusive | share | update 状态。

在将表空间设置为 quiesced exclusive | share | update 之前，要确保它处于 normal 状态：

```
    connect to sample;
    quiesce tablespaces for table staff reset;
    quiesce tablespaces for table staff exclusive; ----------停顿排他
    quiesce tablespaces for table staff share;   ---------停顿共享
    quiesce tablespaces for table staff intent to update; -------停顿意图更新
```

从另一会话执行下列脚本：

```
    connect to sample;
    update staff set salary=50000 where id=60;
    list tablespaces;
```

表空间 USERSPACE1 返回的信息显示，该表空间分别处于 quiesced exclusive| share | update 状态，可执行 quiesce tablespaces for table staff reset 以消除这种状态。

10.3.4　restore pending 和 storage must be defined

在执行了重定向恢复操作的第一部分之后(在发出 SET TABLESPACE CONTAINERS 命令之前)，数据库的表空间就处于这种状态。在使用表空间之前，必须恢复表空间(或是整个数据库)。直到成功完成恢复操作，您才可以连接到数据库。此时，restore_pending 数据库配置参数的值被设为 NO。

当处于 storage may be defined 中的重定向恢复操作的第一部分完成时，所有的表空间都将处于 restore pending 状态。

在将恢复操作重定向到新数据库期间，如果省略设置表空间容器的阶段，或者在设置表空间容器阶段无法获得指定的容器，那么数据库的表空间就会处于这种状态。某些时候会出现后一种情况，例如指定无效的路径名或是磁盘空间不足。

```
backup db sample;
```

假定该备份镜像的时间戳为 20120613204955：

```
restore db sample taken at 20120613204955 into mydb redirect;
set tablespace containers for 2 using (path 'ts2c1');
list tablespaces;
```

LIST TABLESPACES 命令返回的信息显示，表空间 SYSCATSPACE 和 TEMPSPACE1 都处于 storage must be defined、storage may be defined 和 restore pending 状态。storage must be defined 状态比 storage may be defined 状态更重要。

10.3.5　rollforward pending

在对可恢复的数据库执行恢复操作之后，表空间就处于这种状态。在使用表空间之前，必须前滚表空间(或是整个数据库)。如果开启了 LOGARCHMETH1，那么数据库是可恢复的。直到前滚操作成功完成，您才可以激活或连接该数据库。此时，rollfward_pending 数据库配置参数为 NO。

当处于 restore in progress 状态的在线表空间完成恢复操作时，该表空间处于 rollforward pending 状态。

10.3.6　表空间状态总结

表空间状态在某些情况下被用来控制对数据的访问，或者在必要时被用来引出特定用户动作以保护数据库的完整性。除了我们刚才所讲最常看到的几种表空间状态外，我们还会看到 DMS rebalance in progress、backup in progress、load in progress、reorg in progress、restore in progress、rollforward in progress、table space deletion in progress 和 table space creation in progress 等状态。“...in progress”表示表空间处在正在进行某种操作期间的临时状态。如果操作过程出现异常，那么就转变为 pending 状态。这时就需要 DBA 去干预，采用适当的步骤来解除这种暂挂。上面我们举了好多例子，希望大家好好看看，多做练习。

10.4　LOAD 期间表状态总结

DB2 LOAD 实用程序通过表状态(以及锁)来获取对表的访问，并在执行载入操作时维护数据库的一致性。即使载入操作发生异常终止，表状态也将会被保持。您可以用 LOAD QUERY 命令(见图 10-3)确定特定表的状态。LOAD QUERY 命令在运行时检查载入操作的状态，并返回表的状态。如果载入操作完成(或异常终止)了，那么该命令只返回表的状态。

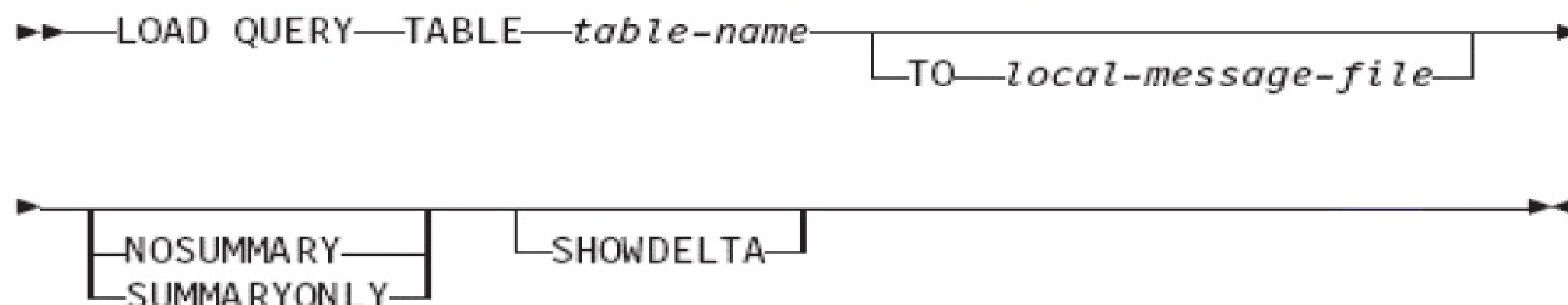


图 10-3　可以使用 LOAD QUERY 命令来确定指定表的状态

虽然在载入操作之前，表所在的表空间不再是停顿的(quiesce 是一种持久性的锁)，但是 load in progress 表空间状态会在执行载入操作时阻止对从属表进行备份。load in progress 表空间状态不同于 load in progress 表状态：所有的载入操作都使用 load in progress 表状态，但是指定 COPY NO 选项的载入操作(针对可恢复的数据库)还是使用 load in progress 表空间状态。

一个表可以同时处于几种状态。例如，如果将数据载入定义了表检查约束的表中，并指定 ALLOW READ ACCESS 选项，那么在执行载入操作期间，该表就处于 check pending、load in progress 和 read access only 状态。

10.4.1　check pending

如果在表上定义了表检查约束，但还未验证新数据与那些已定义约束的兼容性，那么

该表就处于这种状态。例如，DB2 LOAD 实用程序当开始在定义了表检查约束的表上执行载入操作时，就将表的状态设置为 check pending。如果想使该表恢复为 normal 状态，那么需要执行一条 SET INTEGRITY 语句。请看下面的案例：假如载入的输入文件 staff_data.del 拥有以下内容：“11，"Melnyk"，20，"Sales"，10，70000，15000”。执行：

```
    connect to sample;
    alter table staff add constraint max salary check (100000 - salary >0);
    load from staff data.del of del insert into staff;
    load query table staff;
```

LOAD QUERY 命令返回的信息显示，STAFF 表处于 check pending 状态。要想解除这种状态，执行：

```
    set integrity for staff immediate checked;
```

10.4.2 load pending

如果在可提交数据之前，表上的正在执行的载入操作被异常终止，那么该表就处于这种状态。要使表恢复 normal 状态，就需要调用 load terminate、load restart 或 load replace 操作。请看下面的案例：已知载入的输入文件 staffdata.del 拥有大量数据(例如 200 000 或更多条记录)，创建包含载入操作目标表的小型表空间，新建名为 newstaff 的表：

```
    connect to sample;
    create tablespace data space managed by database using (file
'c:\prod\data container1' 256);
    create table newstaff like staff in ts1;
    load from staffdata.del of del insert into newstaff;
    load query table newstaff;
    load from staffdata.del of del terminate into newstaff;
    load query table newstaff;
```

LOAD QUERY 命令返回的信息显示，newstaff 表处于 load pending 状态；在执行 load terminate 操作之后，该表就重新处于 normal 状态。

10.4.3 load in progress

这是一种只在执行载入操作期间才有效的临时状态。当载入操作失败或被中断时，执行 load terminate 或 restart 可以返回正常(normal)状态。

已知载入的输入文件 staffdata.del 拥有大量数据(例如 200 000 或更多条记录)：

```
    update db cfg for sample using LOGARCHMETH1 disk:/logs;
    backup db sample;
```

```
    connect to sample;
    create table newstaff like staff;
    load from staffdata.del of del insert into newstaff copy no;
```

在执行载入操作时，从另一会话执行下列脚本：

```
    connect to sample;
    load query table newstaff;
```

LOAD QUERY 命令返回的信息显示，newstaff 表处于 load in progress 状态。

10.4.4　not load restartable

当执行完前滚操作，接着出现一个失败的载入操作，而该操作未被成功地重新启动或终止时，表就处于这种状态。该表还将处于 load pending 状态。要使该表恢复 normal 状态，就需要执行一条 load terminate 命令。

已知载入的输入文件 staffdata.del 拥有大量数据(例如 20 000 或更多条记录)：

```
    update db cfg for sample using LOGARCHMETH1 disk:/logs;
    backup db sample;
    connect to sample;
    create tablespace data space managed by database using (file
'c:\prod\data container1' 256);
    create table newstaff like staff in ts1;
    connect reset;
    backup db sample;
```

该备份镜像的时间戳为 20120629205935：

```
    connect to sample;
    load from staffdata.del of del insert into newstaff copy yes to
'c:\prod\load backup';
    connect reset;
    restore db sample taken at 20120629205935;
    rollforward db sample to end of logs and stop;
    connect to sample;
    load query table newstaff;
```

LOAD QUERY 命令返回的信息显示，newstaff 表处于 not load restartable 和 load pending 状态。

```
    connect to sample;
    load from staffdata.del of del terminate into newstaff copy yes to
'c:\prod\load_backup';
```

```
load query table newstaff;
```

LOAD QUERY 命令返回的信息显示，newstaff 表现在处于 normal 状态。

10.4.5 read access only

在执行载入操作时，如果指定 ALLOW READ ACCESS 选项，那么表就处于这种状态。read access only 是一种临时状态，允许其他应用程序和实用程序访问在执行载入操作之前就已存在的数据。已知载入的输入文件 staffdata.del 拥有大量数据(例如 200 000 或更多条记录)：

```
connect to sample;
export to st data.del of del select * from staff;
create table newstaff like staff;
import from st data.del of del insert into newstaff;
load from staffdata.del of del insert into newstaff allow read access;
```

在执行载入操作时，从另一会话执行下列脚本：

```
connect to sample;
load query table newstaff;
select * from newstaff;
```

LOAD QUERY 命令返回的信息显示，newstaff 表处于 read access only 和 load in progress 状态。该查询返回 staff 表导出的内容，以及在执行载入操作之前就存在于 newstaff 表中的数据。

10.4.6 unavailable

当前滚无法恢复的载入操作时，表就处于这种状态；这样的表只能被删除，或者从备份镜像恢复。

已知载入的输入文件 staff_data.del 拥有下列内容：“11，"Melnyk"，20，"Sales"，10，70000，15000”。执行：

```
update db cfg for sample using LOGARCHMETH1 disk:/logs;
backup db sample;
```

该备份镜像的时间戳为 20120629182012：

```
connect to sample;
load from staff data.del of del insert into staff nonrecoverable;
connect reset;
restore db sample taken at 20120629182012;
rollforward db sample to end of logs and stop;
connect to sample;
load query table staff;
```

LOAD QUERY 命令返回的信息显示，staff 表处于 unavailable 状态。

10.5　锁相关问题

10.5.1　锁升级

如果 DB2 数据库中发生锁升级现象，那么会影响数据库的并发性能。我们可以增加 locklist 和 maxlocks 来消除锁升级。但是很多时候单纯地调整这两个参数并不能从根本上消除锁升级。所以更多的是调整业务逻辑，创建最合理的索引，编写最高效的 SQL。使 SQL 语句持有锁的时间尽可能短。对于 X 锁升级来说通常发生在一张表上，S 锁升级则通常发生在多张表上。关于锁升级的详细内容，我们在《DB2 数据库性能调整和优化(第 3 版)》的“第 5 章：锁和并发”中有详细讲解，读者可以参考相关内容。

10.5.2　锁等待问题解决流程

对于数据库中出现的锁等待问题，我们可以参考以下步骤来解决：

(1) 可以执行下列 SQL 语句，找出引起锁等待的 SQL 语句：

```
    select AGENT_ID ,substr(STMT_TEXT,1,100) as statement,STMT_ELAPSED_TIME_MS
from table(SNAPSHOT_STATEMENT('sample',-1)) as B where AGENT_ID in (select
AGENT_ID_HOLDING_LK from table(SNAPSHOT_LOCKWAIT('sample',-1)) as A  order by
LOCK_WAIT_START_TIME ASC FETCH FIRST 20 ROWS ONLY ) order by STMT_ELAPSED_TIME_MS
DESC   --------用你的数据库名称替换 SAMPLE 数据库
```

(2) 对引起锁等待的 SQL 语句，用解释工具分析其执行计划。从执行计划中分析该 SQL 语句的索引是否合理，以及能否对该 SQL 语句进行调优。

(3) 尝试使用 db2advis 工具为引起锁等待的 SQL 语句创建最合理的索引。尝试调优引起锁等待的 SQL 语句。

(4) 如果创建索引和调优 SQL 语句仍然不能解决问题，考虑能否根据业务逻辑选择 UR 隔离级别。

(5) 最后考虑能否对引起锁等待的 SQL 语句关联的表采用数据归档、业务分离等手段。

10.5.3　死锁

死锁是数据库中的一种正常现象，每个数据库中都存在死锁情况。死锁是一种特殊情况的锁等待。死锁问题通常和业务逻辑有关，通常我们可以设置 dlchktime 死锁检测时间间隔，把该参数调小，使 db2dlock 后台死锁检测进程能快速地检测到死锁，然后打破死锁

平衡。关于死锁的详细信息，读者可以参考《DB2 性能调整和优化(第 3 版)》的“第 5 章：锁和并发”中的相关内容。

10.6 CPU 常见问题

SQL 性能太差，并发上来导致 CPU 使用率 100%

因SQL 性能差导致 CPU 高是生产环境中最常见的问题，以下几条是经过实践证明的最有效解决办法：

1) 看活动会话的个数和表扫描最多的表。
2) 找到表扫描最多的表对应的 SQL 语句。
3) 通过 OPTGUIDELINES 来干预执行计划。

有时候，DB2 优化器不一定有你想象的那么智能，太复杂的 SQL 很可能会干扰 DB2 优化器的判断，选择不应该使用的执行计划。这个时候，就需要通过一些办法来纠正这种错误，比如：

1) 运行更细粒度的统计分析。
2) 通过 OPTGUIDELINES 来强制修改执行计划。

所以碰到这种问题以后不要慌张，按照上面的步骤做，一定能够找到解决的办法。

10.7 内存常见问题

10.7.1 bufferpool 设置过大，导致数据库无法启动

一定要根据系统的内存资源情况设置 bufferpool，曾经有个客户把 bufferpool 设置为机器内存的 80%，结果导致数据库无法启动。如何解决呢？最后把数据库的配置参数 database_memory 由 automiac 更改为一个比较小的值，例如 100MB 后才能正常启动。然后再连接数据库，调用 db2 alter bufferpool bp_8k size <num_of_pages>来更改缓冲池大小。

10.7.2 排序溢出

```
已分配的专用排序堆总数                    = 20000000
已分配的共享排序堆总数                    = 0
共享排序堆高水位标记                      = 0
后阈值排序 (共享内存)
                                          = 0
总计排序                                  = 24504
总计排序时间 (毫秒)
```

```
                                    = 653400
排序溢出                            = 3420
活动排序数                          = 324
内存池类型                          = 共享排序堆
```

从上面我们可以看到数据库中有很多排序溢出，排序特别消耗资源，合理地设置排序堆 sortheap 非常重要；所以如果数据库中有大量排序溢出，那么考虑增加 sortheap 大小。也考虑把该参数设置为 automiac，让 DB2 自动决定排序堆的大小。

10.7.3 锁内存不足

如果数据库中报告 SQL0912N，那么表明已经达到数据库的锁定请求的最大数目，因为分配给锁定列表的内存不足。这种情况下可考虑增加锁内存 locklist 的大小。建议在提交其他 SQL 语句前，应用程序应该提交 COMMIT 或 ROLLBACK 语句。更多时候我们要考虑调整应用，例如如果我们一次删除 1000 万条记录，那么肯定会产生大量锁；而如果我们分成 10 次删除，每删除 100 万条记录后 COMMIT 释放锁，那么占用锁的资源就会大大减少。

10.8 latch 问题导致系统性能急剧下降

latch 是 DB2 中关于系统级别的资源的竞争，一般通过 db2pd –latches 可以查看系统中是否有 latch 竞争。如果有 LATCH 竞争，则需要现场进行问题诊断，找到问题的根源，并有针对性地采取措施，一般先通过 stack 分析得出是什么数据库对象上的 latch，然后通过经验或 IBM 网站查到通过扩大某项数据库资源，是否能化解该系统资源上的竞争。所以 latch 是最能体现一名技术人员经验和水平的东西，平时要多学习和积累已有的问题案例。

10.9 备份恢复常见问题

在由备份恢复数据库时，遇到 SQL2542 错误，例如下面的例子：

```
$db2 restore db test
SQL2522N  More than one backup file matches the timestamp value provided for
the backed up database image.
```

此错误一般是由于未能提供正确的时间戳造成的。如果有多个数据库的备份，那么在做数据库恢复时，就需要提供正确的路径和时间戳。如果用 DB2 命令行来执行恢复操作，那么在 Windows NT 操作系统中可参照如下命令：

```
    RESTORE DATABASE SAMPLE FROM D:\backups TAKEN AT 20120910090212
```

此命令中要注意路径和时间戳。时间戳可以通过 LIST HISTORY 命令得到：

```
   LIST HISTORY BACKUP ALL FOR TEST
Op Obj Timestamp+Sequence Type Dev Earliest Log Current Log  Backup ID
-- --- ------------------ ---- --- ------------ ------------ --------------
     B  D  20120910090212001   F    D  S0000028.LOG S0000028.LOG

----------------------------------------------------------------------------
     Contains 4 tablespace(s):

    00001 SYSCATSPACE
    00002 USERSPACE1
    00003 SYSTOOLSPACE
    00004 TEST

----------------------------------------------------------------------------
       Comment: DB2 BACKUP TEST OFFLINE
    Start Time: 20120910090212
      End Time: 20120910090222
        Status: A

----------------------------------------------------------------------------
     EID: 46 Location: /home/db2inst4
```

此命令的输出列出了备份的时间戳加上含有 3 位的数字序列：

```
    时间戳+3 位的数字序列=20120910090222
```

所以，可以在 RESTORE 命令中使用时间戳 20120910090222。如果能有多个备份，那么可以使用 LIST HISTORY 命令显示所有备份记录的信息。

10.10　数据移动常见问题总结

如果我们只是在同一个数据库中进行数据移动，那么这将是非常简单的事情。但是实际生活中我们的数据源可能来自异构数据库，平台、操作系统也不同，数据移动过程中会涉及代码页转换问题、标识列、生成列、大对象、XML、日期格式、空值处理、定界符和 PC/IXF 格式问题。下面我们主要讲解这些内容。

10.10.1　标识列

当表中有标识列时，由于标识列由数据库自动维护，因此导入导出时需要特别注意。

1. 标识列导出注意事项

可使用 EXPORT 实用程序从包含标识列的表中导出数据。但是，标识列会限制您输出文件格式的选择。

如果对导出操作指定的 SELECT 语句的格式为 SELECT * FROM tablename，并且未使用 METHOD 选项，那么支持将标识列属性导出至 IXF 文件。然后可使用 IMPORT 命令的 REPLACE_CREATE 和 CREATE 选项重新创建该表，包括其标识列属性。如果通过包含 GENERATED ALWAYS 类型标识列的表创建已导出 IXF 文件，那么成功导入数据文件的唯一方法就是在导入操作期间指定 identityignore 文件类型修饰符，否则会拒绝所有行(发出 SQL3550W)。

2. 标识列导入(IMPORT)和装载(LOAD)注意事项

无论输入数据是否具有标识列值，都可以使用 IMPORT/LOAD 实用程序将数据导入/装载到包含标识列的表中。

如果未使用与标识相关的文件类型修饰符，那么 IMPORT/LOAD 实用程序会遵循下列规则来工作：

- 如果标识列是 GENERATED ALWAYS 列，那么每当输入文件中的相应行缺少标识列值，或者显式指定了 NULL 值时，会为表行生成标识值。如果对标识列指定了非空值，那么会拒绝该行(SQL3550W)。
- 如果标识列是 GENERATED BY DEFAULT 列，那么 IMPORT/LOAD 实用程序会使用用户提供的值(如果提供了这些值的话)；如果缺少数据或者显式指定了 NULL，那么会生成值。

除了通常对标识列数据类型(SMALLINT、INT、BIGINT 或 DECIMAL)的值执行的验证操作以外，IMPORT/LOAD 实用程序不会对用户提供的标识值执行任何其他的验证操作。不报告重复值。此外，在将数据导入到带有标识列的表中时，不能使用 compound=x 修饰符。

有 3 种文件类型修饰符可用来简化将数据导入到包含标识列的表中的操作：identitymissing 和 identityignore 和 identityoverride，如表 10-1 所示。

表 10-1　用来简化将数据导入到包含标识列的表中的 3 个文件类型修饰符

文件类型修饰符	支持 IMPORT	支持 LOAD	描　述
identityignore	yes	yes	指定忽略导入或载入输入文件中任何标识列的值，并为每一行生成新的标识值
identitymissing	yes	yes	指定导入或载入输入文件不包含任何目标表中标识列的值
identityoverride	no	yes	指定在将数据载入到带有 GENERATED ALWAYS 标识列的表中时，要使用的载入输入文件中标识列的值；任何标识列上值为空(或为 NULL 值)的行都会被拒绝

通过表 10-1 我们可以看到，IMPORT 和 LOAD 都支持 identitymissing 和 identityignore 文件类型修饰符，但是 LOAD 支持 identityoverride 文件修饰符而 IMPORT 不支持。了解了上述文件类型修饰符以后，下面我们举一些例子。

例 10-1　在没有标识列的情况下导入数据。

如果输入数据文件未包含任何标识列值(甚至未包含 NULL 值)，那么 identitymissing 文件类型修饰符可以使您更方便地导入带有标识列的表。例如，考虑使用以下 SQL 语句定义的表：

```
create table table1(c1 char(30),
                   c2 int generated by default as identity,
                   c3 real,
                   c4 char(1))
```

用户可能想要将数据从文件(`import.del`)导入到 TABLE1 中，并且此数据可能是从没有标识列的表中导出的。下面是此类文件的一个示例：

```
Robert, 45.2, J
Mike, 76.9, K
Leo, 23.4, I
```

导入此文件的一种方法是通过 IMPORT 命令显式列示所要导入的列，如下所示：

```
db2 import from import.del of del replace into table1(c1, c3, c4)
```

但是，对于包含许多列的表来说，此语法难以使用并且容易出错。导入该文件的另一种方法是使用 `identitymissing` 文件类型修饰符，如下所示：

```
    db2 import from import.del of del modified by identitymissing replace into
table1
```

例 10-2　在带有标识列的情况下装载数据。

identityignore 文件类型修饰符在某些方面与 identitymissing 相反，它指示 LOAD 实用程序：即使输入数据文件包含标识列数据，也应该忽略该数据，并且应该为每一行生成标识值。例如，用户可能想将以下数据按照例 10-1 中定义从文件(load.del)导入到 TABLE1 中：

```
    Robert, 1, 45.2, J
    Mike, 2, 76.9, K
    Leo, 3, 23.4, I
```

如果用户提供的值 1、2 和 3 未用于标识列，那么该用户可以发出以下 load 命令：

```
    db2 load from import.del of del method P(1, 3, 4) replace into table1 (c1,
c3, c4)
```

同样，如果该表包含许多列，那么此方法可能难以使用并且容易出错。identityignore 文件类型修饰符可以将语法简化为：

```
    db2 load from import.del of del modified by identityignore replace into table1
```

当带有标识列的表导出至 IXF 文件时，可使用 IMPORT 命令的 REPLACE_CREATE 和 CREATE 选项来重新创建该表，包括其标识列属性。如果这种 IXF 文件是从包含类型为 GENERATED ALWAYS 的标识列的表创建的，那么只有通过指定 identityignore 文件类型修饰符才能成功导入数据文件，否则会拒绝所有行(SQL3550W)。

例 10-3　装载包含用户提供的值的数据(identityoverride)。

这个文件类型修饰符 IMPORT 不支持，identityoverride 文件类型修饰符用来将用户提供的值装载到包含 GENERATED ALWAYS 标识列的表中。当从另一数据库系统迁移数据并且必须将表定义为 GENERATED ALWAYS 时，或者在将使用 ROLLFORWARD DATABASE 命令之 DROPPED TABLE RECOVERY 选项恢复的数据装载到表中时，此文件类型修饰符非常有用。当使用了此文件类型修饰符时，将拒绝任何未包含标识列数据(或者包含 NULL 数据)的行(SQL3116W)。还应该注意，使用此文件类型修饰符时，可能会违反 GENERATED ALWAYS 列的唯一性属性。在这种情况下，执行 LOAD TERMINATE 操作，然后接着执行 LOAD INSERT 或 REPLACE 操作。下面我们举个例子。

考虑装载输入文件 custdata.del，其中的内容为“3，"Shrek"”：

```
    create table customers (custno smallint not null generated always as identity
(start with 500, increment by 1), custname varchar(16));  ---该表具有标识列，开
始值是 500，而我们导入的值是 3。
```

```
    load from custdata.del of del modified by identityoverride messages load.msg
insert into customers;
    select * from customers;  ---我们可以看到，由于我们指明了 identityoverride，因
此我们提供的值 3 覆盖了 500。
```

3. LOAD 标识列注意事项

在大多数情况下，LOAD 实用程序无法保证对各行指定标识列值的顺序与这些行在数据文件中的出现顺序相同。由于 LOAD 实用程序以并行方式对标识列值的指定进行管理，因此这些值按任意顺序指定。例外情况如下所示：

- 在单一分区数据库中，当 CPU_PARALLELISM 设置为 1 时，不以并行方式处理行。在此情况下，将按照各行在数据文件参数中的出现顺序来隐式地指定标识列值。
- 在多分区数据库中，如果标识列位于分布键中且存在单分区代理程序(未指定多个分区代理程序或 anyorder 文件类型修饰符)，那么将按照各行在数据文件中的出现顺序来指定标识列值。

将表装载到分区数据库中时，如果该表在数据库的分区键中具有标识列并且未指定 identityoverride 文件类型修饰符，那么不能指定 SAVECOUNT 选项。当分区键中存在标识列并且正在生成标识值时，在至少一个数据库分区上从装载阶段重新启动装载操作时，需要从装载阶段开始时就重新启动整个装载操作，这意味着不可能有任何一致点。

注意：

如果符合下列所有条件，那么不允许执行 LOAD RESTART 操作，应改为发出 LOAD TERMINATE 或 REPLACE 操作。

- 要装入的表位于分区数据库环境中，并且其中包含至少一个标识列，该标识列位于分区键中或由分区键中的生成列引用。
- 未指定 identityoverride 文件类型修饰符。
- 失败的上一个装载操作包括在装载阶段后失败的数据库分区。

10.10.2 生成列

当表中有生成列时(生成列是由数据库自动维护的)，对于导出没有需要特别注意的事项，但是导入时需要特别注意。

1. 生成列 IMPORT/LOAD 注意事项

无论输入数据是否具有生成列值，都可以将数据装载到包含(非标识)生成列的表中。IMPORT/LOAD 实用程序生成列值。

如果未使用任何与生成列相关的文件类型修饰符，IMPORT/LOAD 实用程序就会依照下列规则工作：

- 当数据文件中相应的行缺少生成列的值或提供了 NULL 值时，将创建生成列值。如果为生成列提供了非空值，那么将拒绝该行(SQL3550W)。
- 如果为不可空生成列创建了 NULL 值，那么将拒绝整行数据(SQL0407N)。例如，如果将不可空生成列定义为两个表列之和，但这两个表列在数据文件中包含 NULL 值，那么就会发生这种情况。

表有 3 种相互排斥的方法可用来简化将数据载入到包含生成列的表中的操作：generated-missing、generatedignore 和 generatedoverride 文件类型修饰符，如表 10-2 所示。

表 10-2　简化将数据载入到包含生成列的表中的方法

文件类型修饰符	支持 IMPORT	支持 LOAD	描　述
generatedignore	yes	yes	指定导入或载入输入文件中的任何生成列的值都会被忽略，并且会为每一行生成新的值
generatedmissing	yes	yes	指定导入或载入输入文件中的不包含任何目标表中生成列的值
generatedoverride	no	yes	指定在将数据载入带有 GENERATED ALWAYS 列的表中时，要使用的载入输入文件中生成列的值。如果使用了该文件类型修饰符，您的表在执行载入操作之后，将处于检查暂挂状态，以便给您机会验证新数据的完整性。这里的完整性是指与生成列规范的一致性。如果要使表脱离该状态，且不验证输入值，那么就在执行载入操作之后发出下列命令：set integrity for <table-name> generated column immediate unchecked。如果要解除该表的检查暂挂状态，并且验证输入值，那么就发出下列命令：set integrity for <table-name> immediate checked

通过表 10-2 我们可以看到，IMPORT 和 LOAD 都支持 generatedmissing 和 generatedignore 文件类型修饰符，但是 LOAD 支持 generatedoverride 文件修饰符而 IMPORT 不支持。了解了上述文件类型修饰符以后，下面我们举一些例子。

例 10-4　在没有生成列的情况下载入数据。

如果输入数据文件不包含表中所有生成列的任何值(甚至未包含 NULL 值)，那么

generatedmissing 文件类型修饰符会使您能够更方便地将数据载入到包含生成列的表中。例如，考虑使用以下 SQL 语句定义的表：

```
    create table table1(c1 int,
                        c2 int,
                        g1 int generated always as (c1 + c2),
                        g2 int generated always as (2 * c1),
                        c3 char(1))
```

用户可能想将数据从文件(load.del)载入到 TABLE1 中，此数据可能是从没有任何生成列的表中导出的。下面是此类文件的一个示例：

```
     1, 5, J
     2, 6, K
     3, 7, I
```

导入此文件的一种方法是通过 IMPORT 或 LOAD 命令显式列示所要载入的列，如下所示：

```
    db2 import/load from import.del of del replace into table1(c1, c2, c3)
```

但是，对于包含许多列的表来说，此语法难以使用并且容易出错。另一种载入此文件的方法是使用 generatedmissing 文件类型修饰符，如下所示：

```
    db2 import/load from import.del of del modified by generatedmissing
replace into table1
```

例 10-5 在具有生成列的情况下载入数据。

generatedignore 修饰符在某些方面与 generatedmissing 相反，它向 IMPORT/LOAD 实用程序指示：即使输入数据文件包含所有生成列的数据，也应该忽略该数据，并且应该为每一行生成值。例如，用户可能想将以下数据按照上述定义从文件(load.del)载入 TABLE1 中：

```
     1, 5, 10, 15, J
     2, 6, 11, 16, K
     3, 7, 12, 17, I
```

用户提供的非空值 10、11 和 12(用于 g1)以及 15、16 和 17(用于 g2)导致拒绝该行(SQL 3550W)。为了避免这种情况发生，用户可以发出以下 IMPORT/LOAD 命令：

```
    db2 import/load from import.del of del method P(1, 2, 5)  replace into table1
(c1, c2, c3)
```

同样，如果该表包含许多列，那么此方法可能难以使用并且容易出错。generatedignore 文

件类型修饰符可以将语法简化为：

```
    db2 import/load from import.del of del modified by generatedignore  replace
into table1
```

对于 INSERT_UPDATE，如果生成列同时充当主键并且指定了 generatedignore 文件类型修饰符，那么 IMPORT 命令会采用 generatedignore 文件类型修饰符。IMPORT 命令不会在 UPDATE 的 WHERE 子句中用用户提供的值替换此列。

例 10-6　载入包含用户提供的值的数据。

generatedoverride 修饰符只支持 LOAD 实用程序，用来将用户提供的值装载到包含生成列的表中。当从另一个数据库系统迁移数据，或者在将使用 ROLLFORWARD DATABASE 命令 RECOVER DROPPED TABLE 选项恢复的数据装载到表中时，此文件类型修饰符非常有用。当使用了此文件类型修饰符时，将拒绝任何未包含不可空生成列数据(或者包含 NULL 数据)的行(SQL3116W)。使用此文件类型修饰符时，装载操作完成后将使表处于“设置完整性暂挂”状态。要使该表脱离“设置完整性暂挂”状态，而不验证用户提供的值，请执行以下命令：

```
    SET INTEGRITY FOR table-name GENERATED COLUMN IMMEDIATE   UNCHECKED
```

要使该表脱离“设置完整性暂挂”状态并强制验证用户提供的值，请执行以下命令：

```
    SET INTEGRITY FOR table-name IMMEDIATE CHECKED
```

下面我们举个例子，假如载入输入文件 staffdata.del 的内容为：“"Jack"，500000.00，50000”。

```
     create table newstaff (name varchar(16) not null, salary decimal(9,2), bonus
decimal(9,2) generated always as (salary/10));
    load from staffdata.del of del modified by generatedoverride messages
load.msg insert into newstaff;
    set integrity for newstaff immediate checked;
    select * from newstaff;
```

如果生成列在任何分区、维或分布键中，那么会忽略 generatedoverride 文件类型修饰符并且 LOAD 实用程序会生成值，就像指定了 generatedignore 文件类型修饰符一样。这样做是为了避免用户提供的生成列值与生成列定义相冲突，在这种情况下，会将生成的记录放置在错误的物理位置，例如错误的数据分区、MDC 块或数据库分区。

注意：

如果生成列表达式包含受防护的(FENCED)用户定义函数，那么尝试装载到这样的表

中时 LOAP 操作会失败。但是，通过使用 generatedoverride 文件类型修饰符，可以为这些类型的生成列提供您自己的值。

10.10.3 大对象

LOB 导出注意事项

导出包含大对象(LOB)列的表时，默认操作是对每个 LOB 值导出最多 32 KB，以便将其与列数据的余下部分放在同一文件中。如果要导出超过 32 KB 的 LOB 值，那么应将 LOB 数据写至单独的文件以避免截断。

要指定应将 LOB 写至自己的文件，请使用 lobsinfile 文件类型修饰符。此文件类型修饰符指示 EXPORT 实用程序将 LOB 数据放在 LOBS TO 子句指定的目录中。使用 LOBS TO 或 LOBFILE 会隐式激活 lobsinfile 文件类型修饰符。默认情况下，LOB 值与导出的关系数据将写至同一路径。如果使用 LOBS TO 选项指定了一个或多个路径，那么 EXPORT 实用程序将循环使用这些 LOB 路径，以便将每个成功的 LOB 值写入相应的 LOB 文件。还可使用 LOBFILE 选项对输出 LOB 文件指定名称。如果指定了 LOBFILE 选项，那么 lobfilename 的格式为 lobfilespec.xxx.lob，其中 lobfilespec 是为 LOBFILE 选项指定的值，而 xxx 是 EXPORT 实用程序生成的 LOB 文件的序号。否则，lobfilename 的格式为 exportfilename.xxx.lob，其中 exportfilename 是为 EXPORT 命令指定的已导出输出文件的名称，而 xxx 是 EXPORT 实用程序生成的 LOB 文件的序号。

默认情况下，多个 LOB 将写至单个文件，但也可指定将各个 LOB 存储在不同文件中。EXPORT 实用程序会生成 LOB 位置说明符(LLS)，以允许将多个 LOB 存储在一个文件中。写至导出输出文件的 LLS 是指示 LOB 数据在文件中存储位置的字符串。LLS 的格式为 lobfilename.ext.nnn.mmm/，其中 lobfilename.ext 是包含 LOB 的文件的名称，nnn 是该文件内 LOB 的偏移量(以字节为单位计量)，而 mmm 是 LOB 的长度(以字节为单位计量)。例如，db2exp.001.123.456/的 LLS 表示 LOB 位于文件 db2exp.001 中，以 123 字节的偏移量开始进入文件，并且长度为 456 字节。如果 LLS 中指示的大小为 0，那么 LOB 的长度被视为 0。如果长度为－1，那么 LOB 被视为 NULL，并且忽略偏移量和文件名。

如果不希望个别 LOB 数据并置于同一文件，请使用 lobsinsepfiles 文件类型修饰符以将每个 LOB 写至单独文件。

大对象 LOAD/IMPORT 注意事项

因为 IMPORT/LOAD 实用程序将单个列的大小限制为 32KB，所以载入 LOB 时有一些额外注意事项。

默认情况下，IMPORT /LOAD 实用程序将输入文件中的数据视为要载入列中的数据。

但是，如果大对象(LOB)数据存储在主要输入数据文件中，那么数据大小被限制为 32KB。因此，为避免丢失数据，应将 LOB 数据存储在主要数据文件以外的位置，并且应在载入 LOB 时指定 lobsinfile 文件类型修饰符。

LOBS FROM 子句会隐式激活 lobsinfile。载入数据时，LOBS FROM 子句会将载入数据时用于搜索 LOB 文件的路径列表传递至 IMPORT/LOAD 实用程序。如果未指定 LOBS FROM 选项，那么假定要载入的 LOB 文件与输入关系数据文件位于同一路径中。

指示存储 LOB 数据的位置

载入 LOB 信息时，可以使用 LOB 位置说明符(LLS)将多个 LOB 存储在单个文件中。指定 lobsinfile 后，EXPORT 实用程序会生成 LLS 并将其存储在导出输出文件中，并且 LLS 会指示 LOB 数据的位置。载入使用指定的 lobsinfile 选项修饰的数据时，数据库要求每个对应 LOB 列都有对应的 LLS。如果 LOB 列遇到的不是 LLS，那么数据库会将其视为 LOB 文件，并且将把整个文件作为 LOB 载入。

对于 CREATE 方式下的 IMPORT，可通过使用 LONG IN 子句指定将创建 LOB 数据并将其存储在单独的表空间中。

以下示例显示如何载入 LOB 并将其存储在不同文件的 DEL 文件中：

```
   IMPORT/LOAD FROM inputfile.del OF DEL  LOBS FROM /tmp/data MODIFIED BY
lobsinfile INSERT INTO newtable
```

10.10.4　空值处理

在数据移动中，NULL 值是最难处理的，表 10-3 所示的文件修饰符可以用于处理 NULL 值。

表 10-3　用于处理 NULL 值的文件修饰符

文件修饰符	描　　述
nullindchar= x	指定 x 的值(单个字符)将用于替换空值
striptnulls	当将数据载入变长字段时，将截断任何尾部的 NULL(0x00)字符。如果未指定此选项，将会保留 NULL 不能将此选项与 striptblanks 同时指定。它们是互斥的。此选项会取代过时的 padwithzero 选项
striptblanks	当将数据载入变长字段时，将截断任何尾部的空格。如果未指定此选项，将会保留空格 不能将此选项与 striptnulls 同时指定。它们是互斥的。此选项会取代过时的 t 选项

(续表)

文件修饰符	描　　述
keepblanks	指定在执行导入或装载操作时，要保留 CHAR、VARCHAR、LONG VARCHAR 或 CLOB 等类型列中开头或结尾的空白字符(不包含在字符定界符之中)。如果定义的非空(NOT NULL)列包含一个或多个空白字符，并且这些空白字符表示有效数据，那么该修饰符会极其有用。如果您在导入或装载这些数据时未指定 keepblanks 修饰符，实用工具将试图用空(NULL)值来替换空白字符，但因为该列不可为空，所以实用工具会返回错误。请注意，在载入 CHAR 列时，总是会在结尾的空白中填入该列长度。但是，keepblanks 修饰符对于保留 CHAR 列中开头的空白是有必要的

下面我们举一个使用这些文件修饰符的例子。

例 10-7　nullindchar 文件修饰符使用示例。

对于 IXF 格式的文件来说，载入空值非常方便，因为里面已经记录了空值的信息。但是，对于 ASC 格式文件就有一定的难度了，因为 DB2 会直接插入空格，而不是空值。为此，DB2 提供了一个文件修饰符进行控制——NULL INDICATORS。

考虑载入数据文件 null.txt 的内容为——test++++++++ N+++(+表示空格)。

```
    C:\>db2 create table names (firstname varchar(12), lastname varchar(12))
    DB20000I  SQL 命令成功完成。
    C:\> db2 import from null.txt of asc modified by nullindchar=N method l (1
12,13 24) null indicators(0,13) insert into names
    读取行数          = 1
    跳过行数          = 0
    插入行数          = 1
    更新行数          = 0
    拒绝行数          = 0
    落实行数          = 1
    C:\>db2 select * from names
    FIRSTNAME    LASTNAME
    ------------ ------------
    test  -
      1 条记录已选择。
```

上述 IMPORT 语句表示在 13 字符开始的列值为 N 的时候替换为空值。

striptnulls

考虑载入输入文件 orgdata.asc 的内容为：++++++10Head Office++++++++160Corporate +New York+----。

```
    db2 create table orgtemp like org
    db2 load from orgdata.asc of asc modified by striptnulls method l (1 8, 9
22, 23 30, 31 40, 41 53) messages load.msg insert into orgtemp
    读取行数          = 1
    跳过行数          = 0
    装入行数          = 1
    拒绝行数          = 0
    删除行数          = 0
    落实行数          = 1
    db2 select location concat division from orgtemp
    1
    -----------------------
    New York Corporate
```

在上面的例子中，我们看到在使用 **striptnulls** 之后，将输入文件尾部的 **null** 删除了。

keepblanks

```
    db2 create table names (lastname char(16), firstname char(16))
     DB20000I  SQL 命令成功完成。
    db2 create table newnames like names
    DB20000I  SQL 命令成功完成。
    db2 insert into names values (' Tallerico ', ' Teresa ')
    DB20000I  SQL 命令成功完成。
    db2 export to names.del of del modified by nochardel messages export.msg
select * from names
    导出的行数：1
    db2 load from names.del of del modified by keepblanks messages load.msg insert
into newnames
    读取行数          = 1
    跳过行数          = 0
    装入行数          = 1
    拒绝行数          = 0
    删除行数          = 0
    落实行数          = 1
    db2 select firstname concat lastname from newnames
    1
    --------------------------------
     Teresa          Tallerico
      1 条记录已选择。
```

在上述例子中，我们看到在使用 **keepblanks** 之后，导入的数据中的空格都保留了下来。

striptblanks

考虑载入输入文件 orgdata.asc 的内容为：++++++10Head Office++++++++160Corporate +New York+++++。

```
    db2 create table orgtemp like org
    DB20000I  SQL 命令成功完成。
    db2 load from orgdata.asc of asc modified by striptblanks method l (1 8,
9 22, 23 30, 31 40, 41 53) messages load.msg insert into orgtemp
    读取行数          = 1
    跳过行数          = 0
    装入行数          = 1
    拒绝行数          = 0
    删除行数          = 0
    落实行数          = 1
    db2 select location concat division from orgtemp
    1
    -----------------------
    New YorkCorporate
```

在上述例子中，我们看到在使用 striptblanks 之后，头部和尾部的所有空格都被删除了。

10.10.5　定界符注意问题

1. 关于定界符格式，移动数据时的注意事项

移动定界 ASCII(DEL)文件时，一定要确保移动的数据不会因为定界字符识别问题而导致无意中发生改变。为帮助避免发生这些错误，DB2 会强制实施若干限制并提供了许多文件类型修饰符。有许多限制可帮助避免所选定界字符被视为移动数据的一部分。首先，定界符是互斥的。其次，定界符不能是二进制零、换行符、回车符或空格。而且默认小数点(.)不能是字符串定界符。最后，在 DBCS 环境中，不支持竖线(|)字符定界符。下面是一些定界符的使用限制：

- 空格(X'20')永远不能作为有效定界符。
- 在导入期间将删除第一个字符前面的空格或者单元格值中最后一个字符后面的空格。不会删除单元格值中间嵌入的空格。
- 句点(.)由于与时间戳记值中的句点冲突，因此不是有效的字符串定界符。
- 对于纯 DBCS(图形)、混合 DBCS 和 EUC 来说，定界符的范围是 x00 到 x3F。
- 对于使用 EBCDIC 代码页指定的 DEL 数据来说，定界符可能与 shift-in 和 shift-out DBCS 字符不一致。

- 在 Windows 操作系统上，字符定界符外部第一次出现的文件结束符(X'1A')指示文件结束。不会导入任何后续数据。
- 空值表示通常应该赋值的单元格缺少单元格值，也可以表示一串空格。
- 由于某些产品将字符字段的长度限制为 254 或 255 个字节，因此每当选择导出最大长度超过 254 个字节的字符列时，EXPORT 实用程序就会生成警告消息。IMPORT 实用程序可以接受与最长的 LONG VARCHAR 和 LONG VARGRAPHIC 列等长的字段。
- 如果服务器的代码页与客户机的代码页不同，那么建议指定非默认定界符的十六进制表示法。例如：

```
db2 load from ... modified by chardel0x0C coldelX1e ...
```

2. 数据移动期间的定界符问题

双字符定界符

默认情况下，对于 DEL 文件的基于字符的字段来说，字段中的任何字符定界符实例都用双字符定界符表示。例如，假定字符定界符是双引号，如果导出文本“I am 6" tall.”，那么 DEL 文件中的输出文本显示为“"I am 6"" tall."”相反，如果 DEL 文件中的输入文本为“"What a ""nice"" day!"”，那么导入的文本为“What a "nice" day!”。

nodoubledel

可通过指定 nodoubledel 文件类型修饰符来对 IMPORT、EXPORT 和 LOAD 实用程序禁用双字符定界符行为。但要注意的是，双字符定界符行为是为了避免解析错误。对导出使用 nodoubledel 时，字符定界符出现在字符字段中时不会显示为双字符。对导入和装载使用 nodoubledel 时，双字符定界符不会解释为字符定界符的字面值实例。

nochardel

对导出使用 nochardel 文件类型修饰符时，字符字段不会用字符定界符括起来。对导入和装载使用 nochardel 时，字符定界符不会被视为特殊字符并且会解释为实际数据。

```
    db2 create table stafftemp like staff
     DB20000I  SQL 命令成功完成。
    db2 export to staffdata.del of del modified by nochardel messages export.msg
select * from staff
    导出的行数：35
    db2 import from staffdata.del of del modified by nochardel messages
import.msg insert into stafftemp
```

```
读取行数        = 35
跳过行数        = 0
插入行数        = 35
更新行数        = 0
拒绝行数        = 0
落实行数        = 35
```

上述脚本中，staffdata.del 文件的内容如下所示：

```
10,Sanders,20,Mgr  ,7,+98357.50,
20,Pernal,20,Sales,8,+78171.25,+00612.45
30,Marenghi,38,Mgr  ,5,+77506.75,
40,O'Brien,38,Sales,6,+78006.00,+00846.55
50,Hanes,15,Mgr  ,10,+80659.80,
60,Quigley,38,Sales,,+66808.30,+00650.25
70,Rothman,15,Sales,7,+76502.83,+01152.00
```

如果不使用 nochardel 文件修饰符，那么导出的文件内容如下所示：

```
10,"Sanders",20,"Mgr  ",7,+98357.50,
20,"Pernal",20,"Sales",8,+78171.25,+00612.45
30,"Marenghi",38,"Mgr  ",5,+77506.75,
40,"O'Brien",38,"Sales",6,+78006.00,+00846.55
50,"Hanes",15,"Mgr  ",10,+80659.80,
60,"Quigley",38,"Sales",,+66808.30,+00650.25
70,"Rothman",15,"Sales",7,+76502.83,+01152.00
```

chardel

可使用其他文件类型修饰符，以通过手动方式避免默认定界符与数据之间混淆。chardel 文件类型修饰符将单字符 x 指定为要使用的字符串定界符，以代替默认情况下使用的双引号。

```
db2 export to orgdata.del of del modified by chardel'' messages export.msg
select * from org
```

coldel x

同样，要避免将默认情况下使用的逗号用作列定界符，可使用 coldel，它会将单字符 x 指定为列数据定界符。

```
db2 create table orgtemp like org
 DB20000I  SQL 命令成功完成。
db2 export to orgdata.del of del modified by coldel; messages export.msg
select * from org;
```

```
    导出的行数：35
    db2 import from orgdata.del of del modified by coldel; messages import.msg
insert into orgtemp;
    读取行数          = 35
    跳过行数          = 0s
    插入行数          = 35
    更新行数          = 0
    拒绝行数          = 0
    落实行数          = 35
```

delprioritychar

移动 DEL 文件时另一个需要注意的问题就是保留定界符的正确优先顺序。定界符的默认优先级为：行、字符、列。但是，某些应用程序依赖于以下优先级：字符、行、列。例如，如果使用默认优先级，那么 DEL 数据文件：

```
    "Vincent <row delimiter> is a manager",<row delimiter>
```

将解释为两行："Vincent" 和 "is a manager"。原因是<row delimiter>优先于字符定界符(")。如果使用 delprioritychar，那么字符定界符(")优先于行定界符(<row delimiter>)，这意味着同一 DEL 文件将正确地解释为一行："Vincent is a manager"。

假设载入输入文件 contactsdata.del 的内容为："Tallerico","123 Anyplace Street Ourtown H0H 0H0"。

```
    db2 create table contacts (lastname varchar(16), address varchar(64))
     DB20000I  SQL 命令成功完成。
    db2 load from contactsdata.del of del modified by delprioritychar messages
load.msg insert into contacts
    读取行数          = 1
    跳过行数          = 0
    装入行数          = 1
    拒绝行数          = 0
    删除行数          = 0
    落实行数          = 1
    db2 select * from contacts
    LASTNAME         ADDRESS
    -------------- ----------------------------------------------------------------
    Tallerico        123 Anyplace Street Ourtown H0H 0H0
```

10.10.6　PC/IXF 注意问题

PC/IXF 格式注意事项

典型的导出操作包括插入或装载到现有表中的所选数据的输出。但是，也可导出整个

表，以便以后使用 IMPORT 实用程序重新创建。

要导出表，必须指定 PC/IXF 文件格式。然后可通过 CREATE 方式使用 IMPORT 实用程序以重新创建已保存表(包括其索引)。但是，如果出现下列任一情况，那么某些信息不会保存至已导出的 IXF 文件中：

- 索引列名包含十六进制值 0x2B 或 0x2D
- 该表包含 XML 列
- 该表是多维集群表(MDC)
- 该表包含表分区键
- 由于代码页转换而导致索引名长度超过 128 个字节
- 该表是受保护的
- EXPORT 命令包含 SELECT * FROM *tablename* 以外的操作字符串
- 对 EXPORT 实用程序指定了 METHODN 参数

注意：
不建议使用导入的 CREATE 方式。请使用 db2look 实用程序来捕获并重新创建表。

索引信息

如果索引中指定的列名包含“-”或“+”字符，那么不会收集索引信息，并且将返回警告 SQL27984W。EXPORT 实用程序完成处理，并且不会影响已导出的数据。但是，索引信息未保存在 IXF 文件中。因此，您必须使用 db2look 实用程序来单独创建索引。

空间局限性

如果导出的数据超过创建导出文件所在文件系统的可用空间量，那么导出操作会失败。在这种情况下，应该通过在 WHERE 子句中指定条件来对选择的数据量进行限制，以使已导出文件能够存放在目标文件系统中。可以多次调用 EXPORT 实用程序以导出所有数据。表 10-4 列示了可用于 PC/ IXF 文件类型的文件类型修饰符。

表 10-4　可用于 PC/IXF 文件类型的文件类型修饰符

修　饰　符	IMPORT	LOAD	描　　述
indexixf	yes	no	指定导入实用工具将删除表上定义的所有索引，并由 PC/IXF 文件中的索引定义创建新的。该文件类型修饰符只能在表中内容要被替换时使用
nochecklengths	yes	yes	指定即使输入数据超出了目标表列的大小，也应该尝试导入或装载每一行。如果知道输入数据将适合所有情况，那么可以使用该文件类型修饰符

indexixf 示例：

```
db2 create table newemp like employee
DB20000I  SQL 命令成功完成。
db2 export to empdata.ixf of ixf messages export.msg select * from employee
导出的行数：42
db2 import from empdata.ixf of ixf modified by indexixf messages import.msg
replace create into newemp
读取行数          = 42
跳过行数          = 0
插入行数          = 42
更新行数          = 0
拒绝行数          = 0
落实行数          = 42
```

nochecklengths 示例：

```
db2 create table resumes like emp resume
 DB20000I  SQL 命令成功完成。
db2 export to emp resumedata.ixf of ixf messages export.msg select * from
emp resume
导出的行数：8
db2 load from emp resumedata.ixf of ixf modified by nochecklengths messages
load.msg insert into resumes
读取行数          = 8
跳过行数          = 0
装入行数          = 8
拒绝行数          = 0
删除行数          = 0
落实行数          = 8
```

10.10.7 代码页不同注意事项

如果在导入或装载数据时，表的数据页和输入文件的数据页不匹配，那么数据将无法导入，而且会报 SQL0332N 错误。在这种情况下，我们可以使用下面的文件类型修饰符：

forcein 文件类型修饰符

forcein 文件类型修饰符允许导入 PC/IXF 文件，指定导入或装载实用工具不会由于代码页不匹配而拒绝数据，并取消代码页之间的转换。该修饰符在使用时必须谨慎，但是当处理包含了其他情况下无法导入或装载的数据类型或值时，该文件类型修饰符十分有用。此选项不会理会 PC/IXF 文件与目标数据库中数据之间的代码页差别，允许更加灵活地定

义兼容列。

```
    db2 create table resumes like emp resume
    DB20000I  SQL 命令成功完成。
    db2 export to emp_resumedata.ixf of ixf messages export.msg select * from
emp resume
    导出的行数：8
    db2 load from emp_resumedata.ixf of ixf modified by forcein messages load.msg
insert into resumes;
    读取行数          = 8
    跳过行数          = 0
    装入行数          = 8
    拒绝行数          = 0
    删除行数          = 0
    落实行数          = 8
```

codepage 文件类型修饰符

指定 ASCII 字符串用于表示要导入或装载数据的源代码页。如果需要在运行于不同代码页的系统之间移动数据时避免讹误，那么该文件类型修饰符十分有用。首先会将输入文件中的字符数据从由该文件类型修饰符指定的代码页转换成当前的系统代码页，然后从当前系统代码页转换成数据库代码页。请记住，如果输入文件包含当前系统代码页不能识别的字符，那么就无法将该字符导入或装载到数据库中。而且，在代码页转换中占用空间变大的数据可能会被截掉。

```
    db2 create table stafftemp like staff
     DB20000I  SQL 命令成功完成。
    db2 export to staffdata.del of del messages export.msg select * from staff
    导出的行数：35
    db2 load from staffdata.del of del modified by codepage=850 messages load.msg
insert into stafftemp
    读取行数          = 35
    跳过行数          = 0
    插入行数          = 35
    更新行数          = 0
    拒绝行数          = 0
    落实行数          = 35
```

10.10.8　日期格式

在不同的平台和系统上进行数据移动时，日期格式容易出问题。下面我们讲解几个和

日期格式有关的文件类型修饰符，这几个文件类型修饰符仅适用于 ASC 和 DEL 格式。

dateformat

dateformat=" x"可指定字符串，用于表示导入或装载数据的日期格式。每个未指定的元素将被赋值为 1。

考虑导入数据文件 salesdata.asc 的内容为“23.04.2004LUCCHESSI++++++Ontario-South+++++++++++10 2”：

```
    db2 create table salestemp like sales
    DB20000I SQL 命令成功完成。
    db2 import from salesdata.asc of asc modified by dateformat="DD.MM.YYYY"
method l (1 10, 11 25, 26 40, 41 51) messages import.msg insert into salestemp
    读取行数         = 1
    跳过行数         = 0
    插入行数         = 1
    更新行数         = 0
    拒绝行数         = 0
    落实行数         = 1
```

datesiso

指定以 ISO 格式导出所有日期格式：

```
    db2 export to salesdata.del of del modified by datesiso messages export.msg
select * from sales
```

timeformat

timeformat="x"可指定字符串，用于表示导入或装载数据的时间格式。每个未指定的元素将被赋值为 0。

考虑导入输入文件 timedata.asc 的内容为“10.56 PM”：

```
    db2 create table times (timeofday time)
     DB20000I  SQL 命令成功完成。
    db2 load from timedata.asc of asc modified by timeformat="HH.MM TT" method
l (1 8) messages load.msg insert into times
    读取行数         = 1
    跳过行数         = 0
    插入行数         = 1
    更新行数         = 0
    拒绝行数         = 0
    落实行数         = 1
```

timestampformat

timestampformat="*x*"可指定字符串，用于表示导入或装载数据的时间戳格式。在指定 month 和 minute 元素时，一定要避免二义性，因为两者都使用了字母“m”。month 元素必须邻接其他日期元素，而 minute 元素必须邻接其他时间元素：

```
    db2 create table in traytemp like in tray
     DB20000I  SQL 命令成功完成。
    db2 insert into in tray values ('2008-04-16-17.12.30.000000', 'db2inst1',
'Any subject', 'Any note text')
    DB20000I  SQL 命令成功完成。
    db2 export to in traydata.del of del modified by timestampformat="yyyy.mm.dd
hh:mm tt" messages export.msg select * from in tray
    导出的行数：4
    db2 load from in traydata.del of del modified by timestampformat="yyyy.mm.dd
hh:mm tt" messages load.msg insert into in traytemp
    读取行数        = 4
    跳过行数        = 0
    插入行数        = 4
    更新行数        = 0
    拒绝行数        = 0
    落实行数        = 4
```

10.10.9 XML 问题

随着 DB2 V9.1 中引入本地 XML 支持，导出(EXPORT)实用程序也被扩展以支持 XML。如果没有指定任何与 XML 相关的选项而导出表(用 XML 数据定义的)，那么相关的 XML 数据将被写入到与导出的其他关系数据分开的一个或多个文件中。让我们看一个例子。下面的 EXPORT 命令是在 PRODUCT 表上发出的，该表中定义了一个 XML 列：

```
    EXPORT TO prodexport.del DEL MESSAGES msg.out SELECT * FROM product
```

在这个例子中，导出实用程序将生成两个输出文件。其中一个输出文件是 prodexport.del(如图 10-4 所示)，该文件将包含表中的关系数据和 XML Data Specifier(XDS)。

```
File:prodexport.del
"100-100-01",,,,,,"<XDS FIL='prodexport.del.001.xml'OFF='0' LEN''252'/>"
"100-101-01",,,,,,"<XDS FIL='prodexport.del.001.xml'OFF='252' LEN''271'/>"
"100-103-01",,,,,,"<XDS FIL='prodexport.del.001.xml'OFF='523' LEN''304'/>"
"100-201-01",,,,,,"<XDS FIL='prodexport.del.001.xml'OFF='827' LEN''222'/>"
```

图 10-4　输出文件 prodexport.del 的内容

XDS 是用名为"XDS"的 XML 标记表示的字符串。它具有一些属性，用于描述关于列

中实际的 XML 数据的信息。下面是 XDS 字符串中可能出现的一些属性：

- FIL：指定包含 XML 数据的文件的文件名。
- OFF：指定 XML 数据在 FIL 属性指定的文件中的字节偏移量。
- LEN：指定 FIL 属性中指定的文件中的 XML 数据的字节长度。
- SCH：指定用于验证 XML 文档的 XML 模式的全限定 SQL 标识符。稍后将讨论这个属性。

从图 10-5 所示的prodexport.del的内容中可以看出，第一个 XML 数据存储在 prodexport.del.001.xml 中，从 0 字节偏移位置开始，长度为 252 个字节。

在这个例子中，导出实用程序生成的另一个文件是 prodexport.del.001.xml，该文件包含 XML 内容。导出的每个 XML 数据都被连接在一起写入到这个文件中。

File: prodexport.del.001.xml

```
<?xml version="1.0" encoding="UTF-8" ?><product pid="100-100-01"><description><n
ame>Snow Shovel, Basic 22"</name><details>Basic Snow Shovel, 22" wide, straight
handle with D-Grip</details><price>9.99</price><weight>1 kg</weight></descriptio
n></product><?xml version="1.0" encoding="UTF-8" ?><product pid="100-101-01"><de
scription><name>Snow Shovel, Deluxe 24"</name><details>A Deluxe Snow Shovel, 24
inches wide, ergonomic curved handle with D-Grip</details><price>19.99</price><w
eight>2 kg</weight></description></product><?xml version="1.0" encoding="UTF-8"
?><product pid="100-103-01"><description><name>Snow Shovel, Super Deluxe 26" Wid
e</name><details>Super Deluxe Snow Shovel, 26" wide, ergonomic battery heated cu
rved handle with upgraded D-Grip</details><price>49.99</price><weight>3 kg</weig
ht></description></product><?xml version="1.0" encoding="UTF-8" ?><product pid="
100-201-01"><description><name>Ice Scraper, Windshield 4" Wide</name><details>Ba
sic Ice Scraper 4" wide, foam handle</details><price>3.99</price></description><
/product>
```

图 10-5　输出文件 prodexport.del.001.xml 的内容

使用 XML 选项和修饰符导出 XML 数据

与导出大型对象一样，您可以指定被导出 XML 文档的存储路径，还可以指定输出文件的文件名。考虑下面的例子：

```
    EXPORT TO prodexport.del DELXMLTO d:\xmlpath XMLFILE proddesc MODIFIED BY
XMLINSEPFILES XMLNODECLARATION XMLCHAR XMLSAVESCHEMA MESSAGES msg.out SELECT
* FROM PRODUCT
```

在这个例子中，PRODUCT 表的关系数据被导出到 prodexport.del 文件中。然后，所有 XML 数据都被写入到 XML TO 子句指定的目录 d:\xmlpath 中。包含 XML 数据的文件被命名为 proddesc.ext.xml，其中 ext 是序列号(例如 proddesc.001.xml、proddesc.002.xml、proddesc.003.xml 等)。基本文件名是用 XMLFILE 选项定义的。

您也许还注意到，这个例子中使用了一些修饰符。下面对所有与 XML 相关的修饰符作出总结：

- XMLINSEPFILES：导致导出实用程序将导出的每个 XML 文档写入到不同的 XML 文件中。
- XMLNODECLARATION：表明导出 XML 数据时无须使用 XML 声明标记。默认情况下，XML 标记被写在 XML 文档的开头，并包括编码属性。
- XMLCHAR：表明 XML 数据以字符码页编码。默认情况下，XML 数据是以 Unicode 编码的。当使用这个修饰符时，使用的是 codepage 文件类型修饰符或应用程序码页。
- XMLGRAPHIC：表明无论是 codepage 文件类型修饰符还是应用程序码页，导出的 XML 数据都将以 UTF-16 码页编码。注意，这个例子中没有使用 XMLGRAPHIC。

我们要介绍的最后一个选项是 XMLSAVESCHEMA。当插入 XML 文档时，可以用 XML 模式对其进行验证。XMLSAVESCHEMA 选项导致导出实用程序还保存用于每个导出的 XML 数据的 XML 模式。该模式的全限定 SQL 标识符将被存储为相应的 XML Data Specifier(XDS)中的一个 SCH 属性。注意，如果没有用 XML 模式验证导出的 XML 文档，或者该模式对象不再存在于数据库中，那么相应的 XDS 中将不包括 SCH 属性。图 10-6 显示了前面的导出例子的结果。

File: prodexport.del

```
"100-100-01",,,,,,"<XDS FIL='proddesc.001.xml' />"
"100-101-01",,,,,,"<XDS FIL='proddesc.002.xml' SCH='DB2INST1.PRODUCT'/>"
"100-103-01",,,,,,"<XDS FIL='proddesc.003.xml' SCH='DB2INST1.PRODUCT'/>"
"100-201-01",,,,,,"<XDS FIL='proddesc.004.xml' />"
```

File: proddesc.001.xml

```
<product pid="100-100-01"><description><name>Snow Shovel, Basic 22"</name><detai
ls>Basic Snow Shovel, 22" wide, straight handle with D-Grip</details><price>9.99
</price><weight>1 kg</weight></description></product>
```

File: proddesc.002.xml

```
<product pid="100-101-01"><description><name>Snow Shovel, Deluxe 24"</name><deta
ils>A Deluxe Snow Shovel, 24 inches wide, ergonomic curved handle with D-Grip</d
etails><price>19.99</price><weight>2 kg</weight></description></product>
```

File: proddesc.003.xml

```
<product pid="100-103-01"><description><name>Snow Shovel, Super Deluxe 26" Wide<
/name><details>Super Deluxe Snow Shovel, 26" wide, ergonomic battery heated curv
ed handle with upgraded D-Grip</details><price>49.99</price><weight>3 kg</weight
></description></product>
```

File: proddesc.004.xml

```
<product pid="100-201-01"><description><name>Ice Scraper, Windshield 4" Wide</na
me><details>Basic Ice Scraper 4" wide, foam handle</details><price>3.99</price><
/description></product>
```

图 10-6　导出结果

10.11　安全常见问题总结

安全性是一个宽泛的主题，可应用于系统架构的各个不同级别。系统管理员必须持续监控他们的系统，确保系统中采取了适当的安全预防措施。安全性可以应用在系统架构的不同级别上。例如，可以通过安装防火墙来防止外部网络对服务器的未经授权的访问。可以使用一些安全网络协议技术，例如 IPSec，以保证网络上计算机间通信信道的安全性。又如，可以施行严格的密码策略，要求用户选择强密码并经常更换密码。

下面我们总结了一些 DB2 安全使用过程中的最佳安全实践，数据库管理员(DBA)和开发人员如果遵循这些最佳实践，可以确保在 DB2 for LUW 中取得最高级别的安全性。这些最佳实践注重于可以通过数据库管理和编程进行控制的一些安全性因素，但是不包括其他可应用于系统中更大范围内的安全技术或策略。此外列出的最佳实践不分先后顺序，它们的重要性相当，都有助于提高 DB2 数据库服务器的总体安全级别。

10.11.1　从 PUBLIC 撤销隐式的权限和特权

DB2 在内部使用名为 PUBLIC 的伪组，对于 PUBLIC 来说，可以为之授予特权，也可以撤销特权。PUBLIC 实际上不是在外部安全设施中定义的组，但通过它可以为 DB2 认证的用户授予特权。

当创建新数据库时，某些数据库权限和特权就会自动授予 PUBLIC，如表 10-5 所示。

表 10-5　创建数据库后被授予 PUBLIC 的权限和特权

权限或特权	描　　述
BINDADD	允许用户在数据库中创建新的包
CREATETAB	允许用户在数据库中创建新的表
CONNECT	允许用户连接到数据库
IMPLICIT_SCHEMA	允许用户在不存在的模式中创建对象(动态地创建模式)
USERSPACE1 上的 USE 特权	允许用户在 USERSPACE1 表空间中创建表或索引
NULLID 模式上的 CREATEIN	允许用户在 NULLID 模式中创建对象
SQLJ 模式上的 CREATEIN	允许用户在 SQLJ 模式中创建对象
SYSPROC 模式中所有函数和过程上的 EXECUTE WITH GRANT 特权	允许用户调用 SQLJ 模式中的存储过程和执行该模式中的函数，并且可以将该许可授给其他用户

(续表)

权限或特权	描　　述
SQLJ 模式中所有过程上的 EXECUTE WITH GRANT 特权	允许用户调用 SYSPROC 模式中的存储过程
NULLID 模式中创建的所有包上的 BIND 和 EXECUTE 特权	允许用户绑定(BIND)和执行(EXECUTE)NULLID 模式中的包
SYSIBM 模式中表上的 SELECT 特权	允许用户查看系统编目表中的信息
SYSCAT 模式中视图上的 SELECT 特权	允许用户查看系统编目视图中的信息
SYSIBMADM 模式中管理视图上的 SELECT 特权	允许用户查看这些管理视图中的信息
SYSSTAT 模式中编目视图上的 SELECT 特权	允许用户查看系统编目视图中的信息
SYSTAT 模式中视图上的 UPDATE 特权	允许用户更新这些系统编目视图中的统计信息

作为一项最佳实践，在创建新的数据库之后，强烈建议您应立即撤销这些被授给 PUBLIC 的隐式特权。

例如，您可以执行例 10-8 中显示的语句来撤销系统编目视图上的特权和其他被授予 PUBLIC 的隐式特权。不过这个示例还不是最完整的。

例 10-8　创建数据库后撤销 PUBLIC 的隐式特权。

```
CREATE DATABASE testdb;
CONNECT TO testdb;
REVOKE BINDADD ON DATABASE FROM PUBLIC;
REVOKE CREATETAB ON DATABASE FROM PUBLIC;
REVOKE CONNECT ON DATABASE FROM PUBLIC;
REVOKE IMPLICIT SCHEMA ON DATABASE FROM PUBLIC;
REVOKE USE OF TABLESPACE USERSPACE1 FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.COLAUTH FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.DBAUTH FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.INDEXAUTH FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.PACKAGEAUTH FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.PASSTHRUAUTH FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.ROUTINEAUTH FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.SCHEMAAUTH FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.SECURITYLABELACCESS FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.SECURITYPOLICYEXEMPTIONS FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.SEQUENCEAUTH FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.SURROGATEAUTHIDSFROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.TABAUTH FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.TBSPACEAUTH FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.XSROBJECTAUTHFROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.AUTHORIZATIONIDS FROM PUBLIC;
```

```
REVOKE SELECT ON TABLE SYSCAT.OBJECTOWNERS FROM PUBLIC;
REVOKE SELECT ON TABLE SYSCAT.PRIVILEGES FROM PUBLIC;
```

从 DB2 V9.1 开始，CREATE DATABASE 命令语法增加了 RESTRICTIVE 选项。如果该命令中包括了 RESTRICTIVE 选项，那么会导致 RESTRICT_ACCESS 数据库配置参数被设置为 YES，同时不自动授予 PUBLIC 任何特权。如果忽略了 RESTRICTIVE 选项，那么 RESTRICT_ACCESS 数据库配置参数被设置为 NO，上述所有特权都将自动授予 PUBLIC。

如果已使用 RESTRICTIVE 选项创建数据库，并且想要检查是否限制了授予 PUBLIC 的许可权，那么可以发出以下查询来验证 PUBLIC 可以访问哪些模式：

```
SELECT DISTINCT OBJECTSCHEMA FROM SYSIBMADM.PRIVILEGES WHERE AUTHID='PUBLIC'
OBJECTSCHEMA
------------
SYSFUNSYSIBMSYSPROC
```

10.11.2　保护系统编目视图

由于系统目录视图描述数据库中的每个对象，因此如果数据库中有敏感数据，那么可能想要限制对它们的访问。假如，如果不想让任何用户知道其他用户有权访问哪些对象，就应考虑限制对系统目录和管理视图中相关权限的编目视图的访问权。这将防止有关用户特权的信息对可访问该数据库的任何人可用。

还应检查对其收集统计信息的列。记录在系统目录中的某些统计信息可能包含环境中敏感信息的数据值。如果这些统计信息包含敏感数据，那么可能希望从 PUBLIC 撤销对 SYSCAT.COLUMNS、SYSSTAT.TABLES 和 SYSCAT.COLDIST 目录视图的 SELECT 特权。

如果希望限制对系统目录视图的访问，那么可以定义视图，让每个用户检索和它自己相关特权的信息。

例如，视图 MYSELECTS 包括每个特定的表的所有者和名称，已将该表的 SELECT 特权直接授予了一个用户：

```
CREATE VIEW MYSELECTS AS   SELECT TABSCHEMA, TABNAME FROM SYSCAT.TABAUTH
          WHERE GRANTEETYPE = 'U' AND GRANTEE = USER   AND SELECTAUTH = 'Y'
```

此语句中的关键字 USER 为当前会话的用户。

如下语句使此视图可供每个授权用户使用：

```
GRANT SELECT ON TABLE MYSELECTS TO PUBLIC
```

最后，应记住要通过发出下列两条语句来撤销对视图和基本表的 SELECT 特权：

```
REVOKE SELECT ON TABLE SYSCAT.TABAUTH FROM PUBLIC
```

```
REVOKE SELECT ON TABLE SYSIBM.SYSTABAUTH FROM PUBLIC
```

作为一项最佳实践，在创建新的数据库之后，强烈建议保护系统编目视图。

10.11.3 创建实例用户并显式指定组

在安装完 DB2 后，我们需要创建实例。而在 Linux/UNIX 上创建实例需要创建和实例同名的用户。我们在创建实例用户时，一定要做好规划，显式地指定实例组。在有的操作系统上，如果创建实例时不指定组，那么操作系统就隐含地把这个用户指定到组中。然后这个实例用户所在的组隐含地具有该实例最高的 SYSADM 权限。这样这个组中所有用户都具有最高的 SYSADM 权限。这非常危险。所以，创建实例用户时必须显式地指定组。下面我们举一个例子。

例如在 AIX 上要创建 myinst 的实例，如果在创建这个 myinst 用户时没有创建组，那么这个 myinst 用户会自动加入到 staff 组中。实例创建后，staff 组就具有这个实例的 SYSADM 权限。我们都知道 staff 中包含操作系统几乎所有用户。这样是非常危险的。

作为一项最佳实践，建议在创建实例时，做好用户组规划，显式指定实例组。

10.11.4 为 SYSxxx_GROUP 参数使用显式值

DB2 定义了超级用户权限层次结构(SYSADM、SYSCTRL、SYSMAINT 和 SYSMON)，每个权限可以执行子集的一项管理操作，例如创建数据库、强制使用用户离开系统、进行数据库备份。与它们相关联的实例级参数(SYSADM_GROUP、SYSCTRL_GROUP、SYSMAIN_GROUP 和 SYSMON_GROUP)用于控制哪些用户可以继承哪些权限。

每个参数可以被设置为拥有该权限的一个用户组(在外部安全设施中定义)的名称。设置好之后，指定组中的所有用户就继承了该权限。

例如，假设有名为 DBAGRP1 的操作系统组，所有 DBA 用户都是这个组的成员。如果使用例 10-9 中所示的命令将 SYSADM_GROUP 实例参数的值设为 DBAGRP1，那么这个组中的所有用户都将继承 SYSADM 权限。

例 10-9 更新 SYSADM_GROUP 实例参数。

```
UPDATE DBM CFG USING SYSADM_GROUP dbagrp1
```

作为一项最佳实践，应该将每个实例级权限参数的默认值改为显式的组名，以防止不可控的超级用户访问。

在很多企业中，DBA 扮演着多种角色，因而这些参数可以设置为相同的组名。而在大型环境中，由多个 DBA 负责一个系统，因此可以使用不同的组名。除了确保这些参数具有显式值以外，还应该尽量确保参数值所指定的组中的所有用户都确实有必要成为这个组的成员。如果没有这个必要，那么应该从这个组中删除。由于用户和组账户的管理是在 DB2

之外处理的，因此 DB2 不会仔细检查用户应不应该成为组的成员。

10.11.5　跟踪隐式特权

如前所述，创建新的数据库时，PUBLIC 被隐式地授予一些特权。实际上，并不是只有此时才会授予隐式特权。在某些情况下，当用户创建数据库对象(例如表或包)或授予 DBADM 权限级别时，数据库管理器会隐式地将一些特权授给用户。理解被隐式授予的特权有哪些，以及这些隐式特权所蕴涵的安全意义(表 10-6 所示)，这一点非常重要。

表 10-6　授予不同动作的隐式特权

动　作	授予执行该动作的用户的隐式特权
创建新的数据库	将 GRANT of DBADM 权限以及 BINDADD、CONNECT、CREATETAB、CREATE_EXTERNAL_ROUTINE、CREATE_NOT_FENCED_ROUTINE、IMPLICIT_SCHEMA、LOAD 和 QUIESCE_CONNECT 权限授予创建者(SYSADM 或 SYSCTRL) 将 GRANT of BINDADD、CREATETAB、CONNECT 和 IMPLICIT_SCHEMA 授予 PUBLIC 将 USERSPACE1 表空间的 USE 特权授予 PUBLIC 将 SYSPROC 模式中所有过程和函数的 EXECUTE WITH GRANT 特权授予 PUBLIC 将 SQLJ 模式中所有过程的 EXECUTE with GRANT 特权授予 PUBLIC 将 NULLID 模式中所有包的 BIND 和 EXECUTE 特权授予 PUBLIC 将 SQLJ 和 NULLID 模式的 CREATEIN 授予 PUBLIC 将 SYSIBM 编目的 SELECT 授予 PUBLIC 将 SYSCAT 编目视图的 SELECT 特权授予 PUBLIC 将 SYSIBMADM 管理视图的 SELECT 特权授予 PUBLIC 将 SYSSTAT 编目视图的 SELECT 特权授予 PUBLIC 将 SYSSTAT 编目视图的 UPDATE 特权授予 PUBLIC
授予 DBADM 权限	将 GRANT of BINDADD、CONNECT、CREATETAB、CREATE_EXTERNAL_ROUTINE、CREATE_NOT_FENCED_ROUTINE、IMPLICIT_SCHEMA、LOAD 和 QUIESCE_CONNECT 授予目标用户
模式	当显式地创建模式时，CREATEIN、ALTERIN、DROPIN 权限被授予创建模式的用户 当隐式地创建模式时，另外还有 CREATEIN 权限被授予 PUBLIC
创建对象(表、索引、包)	将 GRANT of CONTROL 授予对象创建者
创建视图	仅当用户对视图定义中引用的所有表、视图和昵称均有 CONTROL 特权时，才为其授予 Grant of CONTROL 特权

作为一项最佳实践，应该仔细检查和跟踪执行某动作时所授予的隐式特权。如果以后撤销这个动作，那么应撤销任何隐式特权。

例如，假设您一开始将 DBADM 权限授予用户 niuniu，而随后您又决定撤销此权限。为了撤销 niuniu 的 DBADM 权限，可以使用以下语句：

```
REVOKE DBADM ON DATABASE FROM USER niuniu
```

执行该语句之后，niuniu 将不再拥有 DBADM 权限；然而，他仍然拥有数据库上的 GRANT、BINDADD、CONNECT、CREATETAB、CREATE_EXTERNAL_ROUTINE、CREATE_NOT_FENCED_ROUTINE、IMPLICIT_SCHEMA、LOAD 和 QUIESCE_CONNECT 权限，这些权限是在一开始授予 niuniu 权限时隐式地授给该用户的。这些权限需要显式地从 niuniu 那里撤销。

10.11.6　不授予不必要的特权

开发应用程序时，通常开发人员很少一开始就考虑安全性问题。例如，开发人员通常会用超级用户账户(DBADM 或 SYSADM)来开发和测试他们的应用程序，以免在运行代码时不断碰到安全错误消息。

通常，完成了应用程序的开发和测试阶段后，在开发过程中为解决安全错误消息而授予的特权仍然保留在那里，而实际上它们已经没有存在的必要了。

作为一项最佳实践，应仔细检查在安装和配置应用程序的过程中授予每个用户的特权。确保所有被授出的许可和特权确实都是有必要的。

对于不熟悉 DB2 安全模型的开发人员来说，他们往往因为贪图简单而为自己授予所有可用的特权，以避免安全错误消息。您应该确保所有被授予的特权和权限都确实是有必要的。

10.11.7　使用加密的 AUTHENTICATION 模式

在本章的前面我们已经讲过认证机制，身份认证是指使用一种安全机制对用户 ID 和密码进行验证的过程。用户和组的身份认证是在 DB2 外部设施(例如操作系统、域控制器或 Kerberos 安全系统)中进行管理的。实际的身份认证位置由实例参数 AUTHENTICATION 的值来决定。为了防止用户名和密码以及数据通过网络在客户机与服务器之间传输的时候被窃取，建议采用加密的认证方式。表 10-7 总结了各种加密的身份认证选项。

表 10-7　加密的 AUTHENTICATION 模式总结

AUTHENTICATION 模式	描　　述
SERVER_ENCRYPT	规定在服务器上定义的安全设施上进行身份认证。如果在连接尝试期间指定用户 ID 和密码，那么用户 ID 和密码将与服务器上定义的有效用户 ID 和密码组合相比较，以确定是否允许该用户访问实例或数据库。在这种模式下，在网络上传输用户 ID 和密码时，它们将被加密
KRB_SERVER_ENCRYPT	规定服务器接受 KERBEROS 认证或加密的 SERVER 身份认证模式
DATA_ENCRYPT	规定服务器接受加密的 SERVER 身份认证模式和对用户数据进行加密。使用这种身份认证类型时，以下用户数据也将被加密： ● SQL 和 XQuery*语句 ● SQL 程序变量数据 完成对 SQL 或 XQuery*语句的处理后，来自服务器的包括针对数据的描述的输出数据： ● 查询的部分或全部结果集数据 ● large object(LOB)数据流 ● SQLDA 描述
DATA_ENCRYPT_CMP	规定服务器接受 SERVER 身份认证模式和对用户数据进行加密。 这种身份认证类型可以与不支持 DATA_ENCRYPT 身份认证类型的下级产品相兼容。这些产品可以使用 SERVER_ENCRYPT 身份认证类型，而不必加密用户数据。支持 DATA_ENCRYPT 身份认证类型的产品则必须使用这种身份认证类型
GSS_SERVER_ENCRYPT	规定服务器接受基于 GSS API 插件的身份认证或加密的服务器身份认证模式

应该为环境选择什么样的身份认证模式，这是由数据的敏感级别决定的。如果所有数据都是敏感的，那么应该选择 DATA_ENCRYPT 身份认证模式，这种身份认证模式会对客户机和服务器之间传输的很多数据进行加密。如果只有一小部分数据是敏感的，那么可以选择使用 SERVER_ENCRYPT 模式，这样可以保证密码得到加密，而敏感数据则可以通过不同的机制来得到保护。

注意，AUTHENTICATION 参数是在实例级别设置的，这意味着在相同实例中创建的数据库将使用共同的身份认证模式。如果有两个数据库，每个数据库需要不同的身份认证

模式，那么就需要在不同的实例中创建这两个数据库。

10.11.8 使用独立 ID 创建和拥有对象

创建数据库对象时，通过执行 DDL 语句来创建的那个用户 ID 会拥有数据库对象。如果那个用户 ID 后来不用了(例如这位用户离开了公司)，或者该用户不再需要数据库对象上的数据库访问或权限，那么 DBA 必须撤销该用户的特权。这将导致其他有依赖关系的数据库对象或包失效(或者不起作用)。

一旦成功创建数据库对象或程序包，只要对象的创建者或程序包的绑定者继续持有所引用的数据库对象上必要的特权，这个数据库对象或包就被认为是有效的(而不是不起作用)。因此，当对象创建者或包的绑定者的特权被撤销时，包含静态 SQL 语句的对象和包将失效。

这里简要描述一下这个过程：

(1) 在操作系统中创建新的用户 niuniu，并使用户 niuniu 无效，使之不能被使用。

(2) 从所有操作系统组中删除这个用户 ID，并确认已将该用户所属的用户或组的 CONNECT 特权撤销，以确保该用户 ID 不具有 CONNECT 权限。

(3) 当需要创建新的数据库对象时，或者必须执行其他的 DDL 语句时，再通过 GRANT 语句将执行该动作所需的必要特权授给这个新的用户 ID。例如，为了创建表 T1 上的视图，必须将表 T1 上的 SELECT 特权授给新的用户 niuniu：

```
GRANT SELECT ON TABLE T1 TO USER  niuniu
```

将当前会话授权 ID 暂时设置为新的用户 niuniu：

```
SET SESSION_USER = NIUNIU
```

在这个授权 ID 下，创建数据库对象和绑定包。例如，为了创建表 T1 上的视图，可以执行以下语句：

```
CREATE VIEW V1 AS SELECT * FROM T1
```

(4) 创建好所有必需的数据库对象和包之后，使用组成员关系和组特权来控制对所创建的数据库对象和包的访问：

```
GRANT SELECT ON VIEW V1 TO GROUP1
GRANT EXECUTE ON PKG TO GROUP1
```

(5) 完成上述操作之后，执行以下两条语句，将当前会话授权 ID 重新设置成常规授权 ID：

```
SET SESSION USER = SYSTEM USER
或者
SET SESSION_USER = NIUNIU
```

作为一项最佳实践，使用独立的 ID 来创建和拥有对象。

这种方法可以确保使用独立的用户 ID 与创建数据库对象、绑定包和授予特权的角色相关联。随着时间的推移，面对着用户的来来去去，这样将大大简化数据库模式和特权的管理。这种方法虽然比较麻烦，但是可以有效地避免一些安全问题的出现。

10.11.9　使用视图控制数据访问

控制表数据访问的一种常见方法是使用视图。您不必将整个表数据都公开给应用程序用户，而是可以基于表中的部分列创建视图。例如，假设例 10-10 中定义的表包含保险单信息。

例 10-10　包含保险数据的示例表定义。

```
CREATE TABLE INSURANCE (
   CUSTID              INTEGER NOT NULL PRIMARY KEY、
   SALARY              FLOAT、
   RENEWAL MONTH        VARCHAR(3)、----年龄
   SEX                CHAR(1)、
   MARITAL STATUS       CHAR(1)、
   NUM DEPENDENTS       INTEGER、
   YEAR 1ST POLICY      INTEGER、
   NUM CLAIMS          INTEGER、    -----索赔历史
   CYCLES              INTEGER、
   COMMUTE_DIST         FLOAT);
```

假设某家保险公司的代理公司希望可以访问客户数据，以便分析客户数据，为客户提供更合适的产品。然而，假设根据法律该保险公司不能泄漏个人的年龄或他们索赔的金额。为了满足这些需求，可以在这个表上按照例 10-11 所示方式定义视图，该视图中不包括客户的 RENEWAL_MONTH 和索赔历史。

例 10-11　在包含保险数据的表上定义视图。

```
CREATE VIEW ins v1 sis comp 1 AS  SELECT custid、salary、sex、marital status、
num_dependents、year_1st_policy、cycles、commute_dist   FROM insurance);
```

于是，通过这个视图就可以控制对数据的访问，而不必使用基表。使用 GRANT 语句可以控制对视图的访问，以免所有用户都能查看数据。例如，您可以控制谁能对该视图执行 SELECT、INSERT、UPDATE 和 DELETE 操作。

作为一项最佳实践，在您想隐藏表中的部分列或行的时候，应该使用视图来控制对表的访问。底层表定义发生变化时，使用视图还有助于使应用程序不受影响。和表一样，也可以授予视图的特定特权。当然，除了使用视图，我们还可以使用 LBAC。

10.11.10 使用存储过程控制数据访问

控制对表数据进行访问的另一种方法是使用存储过程。存储过程是一组 SQL 语句，这组 SQL 语句形成的逻辑单元用于执行特定的任务。存储过程是在数据服务器上创建和运行的，用于封装一组经常要运行的操作或查询。例如，雇员数据库上的操作(雇用、解雇、升职、查找)可以编写成存储过程，由应用程序来调用，而不是直接编写在应用程序中。存储过程可以带不同的参数和结果来编译和执行，它们可以有输入、输出和输入/输出参数的任意组合。例 10-12 展示的存储过程的根据业绩评分评定雇员新的工资和奖金。

例 10-12　根据业绩评分评定雇员的工资和奖金。

```
    CREATE PROCEDURE UPDATE SALARY (
     IN empNum  CHAR(6)、
     IN rating  SMALLINT)
      LANGUAGE SQL
      BEGIN
        IF rating = 1 THEN
          UPDATE employee  SET salary = salary * 1.10、bonus = 1500 WHERE empno
= empNum;
        ELSE
          UPDATE employee SET salary = salary * 1.05、bonus = 1000 WHERE empno
= empNum;
        END IF;
      END
```

该存储过程接收两个输入参数：雇员号和评分。然后根据给定的评分更新雇员的工资和奖金。对于获得评分“1”的雇员，为他加薪 10%，并提供$1500 的奖金。对于所有其他评分，为雇员加薪 5%，并提供$1000 的奖金。

作为一项最佳实践，应考虑使用存储过程来控制对数据的访问。通过对存储过程的调用，将直接允许对表的访问。因此，限制用户可以在表上执行的动作，也将同时控制什么用户可以调用存储过程。

如今，很多应用程序的数据库层都设计为存储过程模块化。也就是说，所有数据库访问都可通过存储过程调用来执行。想要执行某个事务(例如更新订单或购买某个产品)的应用程序，只需要从应用程序中调用存储过程即可。这种方法的一个附带好处是，所有逻辑都集中放在一个地方，这使得管理和维护更加容易，而且也使其他应用程序可以重用其功

能。这种方法与目前市场上比较流行的面向服务架构(Service Oriented Architecture，SOA)非常吻合。

还可以通过 GRANT 和 REVOKE 语句控制对存储过程的访问。想要调用存储过程的用户需要被授予 EXECUTE 权限。根据绑定选项和 SQL 语句是静态还是动态，存储过程引用到的各个对象可能需要更多的特权。

10.11.11　使用 LBAC 控制数据访问

基于标签的访问控制(Label Based Access Control，LBAC)。LBAC 使你可以决定谁拥有不同行和列上的写访问权限，谁拥有读访问权限。

安全管理员权限(SECADM)被用于配置 LBAC，具体做法是创建安全策略，安全策略实际上定义了用于决定谁可以访问什么数据的标准。创建好安全策略后，安全管理员创建安全标签，安全标签也是安全策略的一部分。标签可以基于任何标准，例如工作名称、用户是否是管理人员或者用户是否属于某个特定的部门。创建好安全标签之后，便可以将安全标签与表中的行和列相关联，以保护其中保存的数据。安全管理员通过为用户授予安全标签来允许用户访问受保护的数据。当用户试图访问受保护的数据时，用户的安全标签将与用于保护该数据的安全标签相比较。

安全管理员还可以为用户授予豁免权(exemption)。豁免权使用户可以访问其安全标签不允许访问的受保护数据。如果用户试图访问受保护的列，而他们的 LBAC 凭证又不允许他们访问该列，那么这样的访问将失败，用户会收到一条错误消息。

作为一项最佳实践，应考虑使用 LBAC 作为控制对敏感数据访问的一种方法。LBAC 很容易配置，您可以对它进行定制以满足特定的安全环境。

使用这种非常易于定制的新安全特性可以让开发人员将精力集中在业务逻辑的开发上，而不用关心太多的安全问题。通过使用 LBAC，我们可以在数据服务器上实现特定的安全需求。

10.11.12　对重要敏感数据进行加密

数据库已经能够阻止未经授权的人看到其中的数据，这通常是通过数据库管理器中的特权和权限来实现的。我们都知道 DBA 对表中的数据有完全访问权限，但是在有些环境下，表中可能还有数据拥有者不希望任何其他人看到的某些信息，例如病人姓名等。特别是基于 Web 的应用程序，这一问题就更加明显了。在这种应用程序中，用户输入的数据(比如信用卡账号)需要保存起来，以备同一用户以后使用该应用程序。同时，用户拥有者希望能够确保任何其他人不能访问这种数据。

为了实现这种功能，DB2 内置了一些 SQL 函数，这些函数允许应用程序加密和解密

数据。当将数据插入到数据库中时，可以使用用户提供的加密密码对其加密。当检索该数据的时候，必须提供相同的密码才能解密数据。对于要多次使用同一密码的情况，可以使用赋值语句设置 ENCRYPTION PASSWORD 值，并令其在某次连接期间内有效。

下面显示了这些常用的加密函数：

- Encrypt(StringDataToEncrypt, PasswordOrPhrase, PasswordHint)
- Decrypt_Char(EncryptedData, PasswordOrPhrase)
- GetHint(EncryptedData)

用于对数据进行加密的算法是 RC2 分组密码(block cipher)，它带有 128 位的密钥。这个 128 位的密钥是通过消息摘要从密码得来的。加密密码与 DB2 认证无关，仅用于数据的加密和解密。这里可以提供可选参数 PasswordHint，这是一个字符串，可以帮助用户记忆用于对数据加密的 PasswordOrPhrase(例如，可以使用'George'作为记忆 'Washington'的提示)。

1. 列级加密

列级加密(column level encryption)意味着对给定列中的所有值都使用相同的密码进行加密。这种类型的加密可以在视图中使用，也可以在使用了公共密码的情况下使用。当对一个或多个表中所有的行使用相同的密钥时，ENCRYPTION PASSWORD 专用寄存器将十分有用。

例 10-13 这个例子使用 ENCRYPTION PASSWORD 值来保存加密密码。它对雇员的社保账号进行加密，并以经过加密的形式存储在 EMP 表中。

```
db2 create table emp(ssn varchar(124) for bit data);
db2 set encryption password = 'Ben123';
db2 insert into emp(ssn) values(encrypt('289-46-8832'));
db2 insert into emp(ssn) values(encrypt('222-46-1904'));
db2 insert into emp(ssn) values(encrypt('765-23-3221'));
db2 select decrypt_char(ssn) from emp;
```

例 10-14 这个例子在结合使用视图的情况下使用 ENCRYPTION PASSWORD 值来保存加密密码。下面的语句声明了 EMP 表的一个视图：

```
create view clear_ssn (ssn) as select decrypt_char(ssn) from emp;
```

在应用程序代码中，我们将 ENCRYPTION PASSWORD 设置为'Ben123'，现在可以使用 clear_ssn 视图了。

```
set encryption password = 'Ben123';
   select ssn from clear_ssn;
```

2. 行-列(单元格)或集合-列级加密

行-列(单元格)或集合-列(Set-Column)级加密意味着在加密数据列内使用多个不同的密码。例如，Web 站点可能需要保存客户信用卡账号(ccn)。在这个数据库中，每个客户可以使用他自己的密码或短语来加密 ccn。

例 10-15　Web 应用程序收集关于客户的用户信息。这种信息包括客户姓名(存储在宿主变量 custname 中)、信用卡账号(存储在宿主变量 cardnum 中)和密码(存储在宿主变量 userpswd 中)。应用程序像下面这样执行客户信息的插入操作：

```
    insert into customer(ccn, name)
values(encrypt(:cardnum, :userpswd), :custname)
```

当应用程序需要重新显示某客户的信用卡信息时，客户要输入密码，同样密码也要存储在宿主变量 userpswd 中。之后，可以像下面这样检索 ccn：

```
    select decrypt_char(ccn, :userpswd) from customer where name = :custname;
```

例 10-16　这个例子使用提示来帮助客户记忆他们的密码。这里使用与例 10-15 相同的应用程序，该应用程序将提示保存到宿主变量 pswdhint 中。假设 userpswd 的值是'Chamonix'，pswdhint 的值是'Ski Holiday'。

```
    insert into customer(ccn, name)
    values(encrypt(:cardnum, :userpswd, :pswdhint), :custname)
```

如果客户请求关于所使用密码的提示，那么可以使用下面的查询：

```
    select gethint(ccn) into :pswdhint from customer where name = :custname;
    pswdhint 的值被设置为"Ski Holiday"。
```

3. 加密非字符值

数值和日期/时间数据类型的加密通过强制类型转换得到间接支持。非字符的 SQL 类型通过强制转换为“varchar”或“char”就可以被加密了。

例 10-17　加密和解密 TIMESTAMP 数据时用到的强制类型转换函数。

```
    -- Create a table to store our encrypted value
    create table etemp(c1 varchar(124) for bit data);
    set encryption password 'next password';
    -- Store encrypted timestamp
    insert into etemp values encrypt(char(CURRENT TIMESTAMP));
    -- Select & decrypt timestamp
    select timestamp(decrypt_char(c1)) from etemp;
```

例 10-18 加密/解密 double 数据。

```
set encryption password 'next password';
insert into etemp values encrypt(char(1.11111002E5));
select double(decrypt_char(c1)) from etemp;
```

作为一项最佳实践，建议对表中的重要敏感数据进行加密。

加密，就其本质而言，会使大部分 SQL 语句慢下来。但是如果多加注意，多加判断，还是可以将大量的额外开销降至最低。而且，加密数据对于数据库的设计来说有着很大的影响。通常，您需要对模式中的一些敏感数据元素进行加密，例如社保账号、信用卡账号、病人姓名等。

10.12 SQL0805 和 SQL0818 错误

SQL0818 错误

SQL0818 错误是由于绑定文件中头部的时间戳和应用程序本身的时间戳及数据库应用程序包内部的时间戳不一致而造成的。

在静态 SQL 嵌入 C/C++编程中，存在 3 个 SQL 时间戳，分别如下：

第 1 个在绑定文件的头部，可以通过 db2bfd 参看：

```
/home/switch$ db2bfd -s -b -v fkapp.bnd
fkapp.bnd:  Header Contents
Header Fields:
Field          Value
-----          -----
releaseNum     0x700
Endian  0x4c
numHvars       2
maxSect        1
numStmt        12
optInternalCnt  4
optCount        9
Name                Value
------------------  -----
Isolation Level     Cursor   Stability
Creator             "ORACLE   "
App Name            "FKAPP    "
Timestamp           "QAqjTJDV:2012/03/09 19:35:42:16"
Cnulreqd            Yes
```

```
Sql Error            No package
Validate             Bind
Date                 Default/local
Time                 Default/local
*** All other options are using default settings as specified by the server***
```

第 2 个时间戳在数据库的系统表中，执行如下命令：

```
db2 select unique_id from syscat.packages where pkgname='FKAPP'
```

得出应用程序的时间戳，如下所示：

```
/home/switch$ db2 select unique id from syscat.packages where
pkgname='FKAPP'
UNIQUE ID
---------
QAqjTJDV
```

第 3 个时间戳包含在程序 fkapp.c 的头部，在程序编译后自动包含在可执行应用程序的头部，所以这 3 个时间戳必须一致，否则就会出现 SQL0818 错误。

在清楚了产生 SQL0818 错误的原因后，下面我们先把数据库中时间戳不一致的应用程序包(package)删除掉：

```
db2 drop package pkgname
```

然后重新绑定生成新的应用程序包 db2 bind *.bnd 即可。

SQL0805 错误

SQL0805N 错误是指应用程序在执行时在数据库中找不到应用程序包"pkgname"。不能完成语句，因为未在 syscat.packages 系统表目录中找到必要的程序包。"<pkgname>"的格式为：

- 'pkgschema.pkgname 0Xcontoken'，其中的一致性标记以十六进制给出。
- 'pkgschema.pkgname.pkgversion'，如果程序包版本为空字符串，那么名称省略'.pkgversion'。
- '%.pkgname'，如果设置了 CURRENT PACKAGE PATH，那么 CURRENT PACKAGE PATH 中模式名的设置是由百分比字符('%')指示的。

造成 SQL0805 错误的可能原因是：

- 程序包未绑定或已删除。
- 如果试图运行 DB2 实用程序，那么 DB2 实用程序可能需要重新绑定至数据库。

- '%.pkgname'，虽然设置了 CURRENT PACKAGE PATH，但是在 CURRENT PACKAGE PATH 的任何模式中都找不到名为'pkgname'的程序包。

注意，当对给定的 package-schema.package-name 使用版本标识时，可能有以相同的程序包模式和程序包名定义的程序包，但是未找到正确的程序包，原因是现有程序包与请求的版本或一致性标记不一致。程序包必须与程序包名的所有 3 个部分相匹配。当正在使用多个版本时，导致出现此消息的附加原因为：

- 正在执行的应用程序的版本已预编译、编译和链接，但是未绑定，或已绑定但是后来删除了该版本的程序包。
- 应用程序已预编译和绑定，但是未编译或链接，所以正在执行的应用程序不是最新的。
- 程序包由与生成编译并链接至应用程序可执行文件的已修改源文件的预编译不同的源文件预编译生成的绑定文件绑定。
- 新应用程序同与现有的程序包相同的名称(和版本)绑定，这样就替换了现有的程序包。如果运行与替换的程序包相关联的应用程序，就会导致此错误。

在所有上述情况下，请求的一致性标记与现有版本的一致性标记不匹配，因此认为未找到程序包。

那么如何解决呢？可以采用以下方法：指定正确的程序包名或绑定该程序。如果正在运行的应用程序未与数据库绑定，就与数据库管理员联系以执行必需的绑定。确保正在执行的应用程序或对象模块是与程序包的预编译和绑定相关联的已编译和链接的修改源代码。

如果设置了 CURRENT PACKAGE PATH，确保在 CURRENT PACKAGE PATH 中指定了包括程序包的模式。

可以使用下列 SQL 语句来查询目录，以确定是否有程序包的不同版本：

```
SELECT PKGSCHEMA, PKGNAME, PKGVERSION, UNIQUE ID  FROM SYSCAT.PACKAGES
  WHERE PKGSCHEMA='pkgschema' and PKGNAME='pkgname'.
```

注意 UNIQUE_ID 列与一致性标记相对应。

如果 DB2 实用程序需要重新绑定至数据库，那么当连接至数据库时，数据库管理员可以通过从实例的 bnd 子目录发出下列 CLP 命令之一来完成此操作：

对于 DB2 实用程序，发出“DB2 bind @db2ubind.lst blocking all grant public”。

对于 CLI，发出“DB2 bind @db2cli.lst blocking all grant public”。